Discrete Event Simulation in Forth

Leonard F. Zettel
Room S3035, SRL
Ford Motor Company
P.O. Box 2053
Dearborn, Michigan 48121, U. S. A.

Abstract

A Forth based discrete event simulation system is presented, with words for scheduling events, executing the simulation, or optionally stepping through it. An example problem, the barbershop simulation, is presented and discussed. Forth is shown to have considerable advantage over implementation in conventional procedural languages.

Introduction

Discrete event simulation is a widespread application of computing. This paper will describe the implementation of a discrete event simulation in Forth and demonstrate that, the characteristics of Forth lead naturally to a system with considerable advantages over systems coded in conventional procedural languages like FORTRAN or Pascal. For a thorough description of a FORTRAN based simulation system see [PRI74]. Discrete event simulation has been used to model and study a great variety of phenomena. It is particularly well suited to the study of problems involving waiting lines—queueing problems. Simulation studies have been made of waiting lines at banks [PRI74], the flow of work through factories [SHE84], and the performance of computer systems [SIG84], to mention just a few examples.

A Simulation Problem

To guide the following discussion, let us consider a simple queueing situation, the barber shop problem [SCH74]. Customers arrive at random times, wanting to get a haircut. There is one barber, and (s)he can serve a typical customer in about sixteen minutes. A customer who arrives when the barber is busy takes a seat in the shop and waits his turn. In more technical language, the customer has joined a single-server, first-in, first-out (FIFO) *queue*. A customer arriving when three or more people are already waiting may *balk*—decide not to join the queue but instead get a haircut at another time or place. Items that might be of interest in studying this system, for different arrival rates of customers and haircut times, could include the proportion of time the barber is busy (his *utilization*), the length of time customers have waited (average, maximum, percentile distribution), the length of the waiting line, and the proportion of balks.

In approaching the discrete event simulation problem we consider that at any time the system can be in only one of a set of *states*. For our purposes the state of the barber shop is completely defined by whether the barber is busy or idle and, if busy, by the number of customers waiting to get haircuts. Transitions from state to state occur instantaneously and are called *events*. For a more rigorous definition of a discrete event simulation system see [SUR83]. For the barbershop system we need two events: arrival of a customer, and completion of a haircut.

The commonest approach to discrete event simulation, and the one we will take here, involves the maintenance of a calendar of events. The simulation proceeds by having the system remove the next most imminent event from the calendar, update a simulation clock to the time of the event removed, change the system state according to the rules for that event (which may be conditional on the state of the system at the time of event occurrence), and possibly add one or more new events to the calendar.

In the barber shop problem, the simulation would be started off by showing no one waiting for a haircut, setting the status of the barber to "idle", and placing a customer arrival event on the calendar. When executed, a customer arrival event would do the following:

1) Schedule the next customer arrival event.

2) If the barber is idle, set the state of the barber to busy and put a haircut completion event on the calendar.

3) If the barber is busy and there are fewer than n customers waiting, increase the number of customers waiting by one.

4) If there are n or more customers waiting, increase the number of balks by one and do nothing else, thus effectively the customer does not enter the barbershop.

The haircut completion event actions are:

1) If there are no customers waiting, set the state of the barber to idle.

2) If there are customers waiting, reduce the number of waiting customers by one and schedule another haircut completion event.

To round out a practical system, there would probably be one or more statistics reporting events and an end-of-simulation event, either scheduled to occur at a definite time (eight hours after the shop opened, say) or triggered by some system statistic like the completion of 100 haircuts.

An event calendar is an instance of a priority queue, the priority in this case being the event occurrence time. Schemes for priority queues and event calendars have included linear lists, indexed lists, heaps and leftist trees. Any of these can be implemented in Forth. For a review of these structures and issues related to their relative performance see [MCC81].

In FORTRAN the event type is usually identified by an integer carried as part of the event notice on the calendar. A particular event is executed by means of a subroutine call after a computed GO TO, with all the overhead that that implies. Space for the event calendar and other files, and variables and arrays for system state representation and statistics collection, are kept in large COMMON blocks. Exceeding any of the storage limits, besides being difficult to detect, necessitates often cumbersome and complicated reallocation of memory. As a result, initial allocations err on the generous side, resulting in the waste of computer resources as we wait for the rare worst case to happen.

Simulation in Forth

Proper implementation in Forth avoids all of this. Variables and arrays representing the system state are naturally global. Data structures that will change size during the simulation can dynamically allocate and free space in a manner that does not interfere with other structures, so that the only limit encountered will be that of the overall space accessible to the Forth dictionary. The key data structure is the event calendar, here a linear linked list [KNU72] with variable node size. We use a linear linked list for simplicity, and because it will give performance equal to or better than the alternatives as long as the event calendar does not grow too long. Figure 1 shows a conceptualization of the list, and Figure 2 represents more nearly the way it would appear in the computer memory allocated to the Forth dictionary.

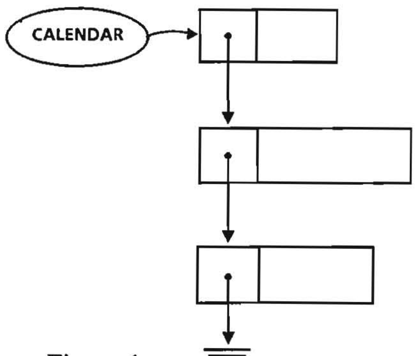

Figure 1.

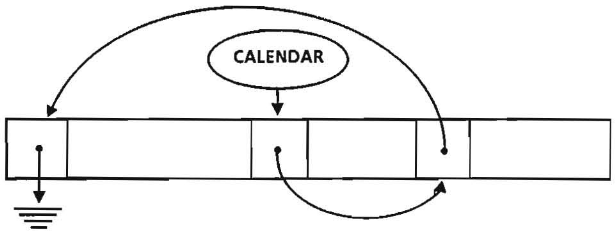

Figure 2.

The variable **CALENDAR** points to the top of the list. The minimum node size for an event notice is five cells. The first cell contains the pointer to the next node on the list, or is zero if this is the last node. Putting the link node first increases the speed with which we can pass from node to node during processing and also simplifies the code of the list processing routines **PUSH** and **POP**, since they then don't have to distinguish between an occurrence of the address of the pointer variable and the occurrence of a node address. The second cell of a node contains the node size in cells, including the pointer cell and node size cell.

The third and fourth cells contain the double number event time. The fifth cell has the compilation address of the routine associated with the event. If the event routine requires parameters, then they are placed in the sixth and succeeding cells of the node. When the event routine executes the parameters will be in order on top of the parameter stack. The event calendar is arranged in order of increasing execution times, so the notice for the next event to be executed is always at the top of the list.

The event time is expressed as an integer, the customary form of number representation in Forth. In recent years it has become fashionable among some in the simulation community to consider a floating point time an advantage, because its reduced granularity would hopefully eliminate problems which can arise when two distinct events are scheduled for the same moment in time. If the event that ends up first on the calendar causes state changes that influence the action of the second event, then results of the simulation will depend on whether it appears first or second. In the opinion of the author, this "advantage" is likely to be more than offset by the changing granularity of a floating point clock as the simulation progresses. At large simulation times the results of events close to each other in time will be modeled with less and less accuracy. Finally, we note that by using a double number simulation time, a time unit of one second (for many simulations a rather fine granularity) lets a simulation span sixty years before clock overflow.

The Forth word **SCHEDULE** (screen 39) inserts a new event notice at the proper spot in the event calendar. **NEXT.EVENT** (screen 40) takes the top event notice off the calendar, updates the simulation clock (kept in the double number variable **CLOCK**), puts any event attributes on the parameter stack, and **EXECUTE**'s the associated event routine. Here we take advantage of the threaded code properties of Forth to minimize the overhead associated with invoking an event routine. If there is no event notice on the calendar, **NEXT.EVENT** aborts with a message to that effect. The continuing execution of the simulation is accomplished by **SIMULATE**, (screen 40). It simply loops endlessly, invoking **NEXT.EVENT** until interrupted by input from the terminal.

At this point note that implementation in Forth has given us an important bonus at next to no cost. By invoking **NEXT.EVENT** directly from the terminal we can step through the simulation event by event, examining or changing any aspect of the system at our leisure, directly from the keyboard. This is a greater degree of interactive capability that that offered by any other simulation system known to the author.

It is fairly common to want an event routine to schedule a future instance of itself, so we include the word **ME** which will put the compilation address of the word in which it is embedded on the stack when executed. The definition of **ME** uses words not found in the Forth-79 standard. The usage here is FIG Forth and follows [PET81]. We note in passing that the commonly used **MYSELF** (**RECURSE** in some implementations) can be defined as follows:

```
: MYSELF [COMPILE] ME EXECUTE ; IMMEDIATE
```

Although the author has not yet encountered the situation in practice, it is conceivable that there might be a closed circle of event invocations in a simulation, i.e. event A schedules an instance of routine B which in turn schedules A. This and its more complicated variations can be handled by any of the vectored execution techniques available to the Forth programmer.

Now let us consider some of the specifics of simulating our example (screens 88-90). We define the Boolean constants **TRUE** and **FALSE** according to the Forth-79 standard (in Forth-83 **TRUE** should be **-1**). Variable **WAITING** holds the number of customers waiting to get a haircut. **BALKS** holds the number of customers who have balked since the simulation started. **BARBER** holds the value **TRUE** when the barber is busy, and **FALSE** if (s)he is idle. The arrival time of the next customer will be determined by a random draw of an integer between **ARRIVAL.MIN** and **ARRIVAL.MIN** plus **ARRIVAL.SPREAD** minus one, each integer having an equal probability of being chosen. **HAIRCUT.MIN** and **HAIRCUT.SPREAD** function the same way for haircut completion times.

For convenience we have included the word **SETUP.BARBERSHOP** to initialize the simulation. It sets the number served, the number who have balked, and the simulation time all to zero. The state of the system is put to the traditional "empty and idle" by putting zero in **WAITING** and **FALSE** in **BARBER**. Finally, **CUSTOMER.ARRIVES** is executed.

CUSTOMER.ARRIVES schedules the occurrence of the next customer arrival. The code for **CHOOSE** and **RANDOM** is taken from [BRO81] (screen 17).

Next, if the barber is busy and less than four people are waiting we add one to **WAITING** and exit. Note that since for haircutting purposes all customers are created equal, in this case there is no need to create an actual file of waiting customers. The number waiting is sufficient for all the purposes of the simulation. If the barber is busy and four or more people are waiting, we add one to **BALKS** and exit. If the barber is idle we change the barber state to busy and schedule **FINISH.HAIRCUT**. Note that in this version of Forth, **["]** puts the parameter field address of the following word on the stack, so we get the compilation address required by **SCHEDULE** by following the event routine name with **CFA**, which replaces the parameter field address with the corresponding compilation address.

In **FINISH.HAIRCUT** we add one to **SERVED**. If a customer is waiting we decrement **WAITING** by one and schedule another haircut completion, otherwise we set the status of **BARBER** to idle and exit. Finally, we include the word **REPORT** to display the current status of the system—number of customers served, number waiting, and number of potential customers who have balked.

Instrumentation and Software

The example shown was run on a Commodore 64™ with a 1541 disk drive and a VIC-1525 graphic printer. The Forth system was Superforth 64™ V 2.2 from Parsec Research, run in its 79-Standard mode. Superforth 64 is an extension of MVP Forth from Mountainview Press. With the exceptions noted in the text, every effort was made to adhere to the Forth-79 standard. The screens shown here use the Brodie baskslash (\, ASCII 92) to indicate that the rest of the line is comments [BRO84]. The actual screens used the Commodore code 92, which prints as an English pound sign (£).

Figure 3 shows some of the output from **LOAD**ing screen 97, including both computer system performance figures and simulation results. An additional 86 bytes not shown was consumed by the event calendar in the course of the simulation.

```
SYSTEM COMPILATION
727  BYTES 34.68 SECONDS

MODEL COMPILATION
601  BYTES 19.50 SECONDS

AFTER 8 HOURS 0 MINUTES 0 SECONDS.
26 CUSTOMERS HAVE BEEN SERVED,
0  ARE WAITING AND
0 HAVE BALKED.
9.03 SECONDS  EXECUTION.

END OF SIMULATION.
```

Figure 3.

Conclusions

We have shown that a discrete event simulation system can be implemented in Forth using very modest amounts of memory and with acceptable run times. In addition, the nature of the Forth language makes the system inherently interactive and extendable, features not found in more conventional simulation systems.

References

[BRO81] Brodie, Leo, *Starting Forth*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[BRO84] Brodie, Leo, *Thinking Forth, A Language and Philosophy for Solving Problems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.

[KNU72] Knuth, Donald E., *The Art of Computer Programming Vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, Mass. 1972.

[MCC81] McCormack, William M. and Robert G. Sargent, "Analysis of Future Event Set Algorithms for Discrete Event Simulation", *Communications of the ACM*, Vol. 24, #12:801-812, Dec. 1981.

[PET81] Petersen, Joel V., "Recursion and the Ackermann Function", *Forth Dimensions*, Vol. III, #3:89-90, 1981.

[PRI74] Pritsker, A. Alan B., *The GASPIV Simulation Language*, John Wiley & Sons, New York, NY 1974.

[SCH74] Schriber, Thomas J., *Simulation Using GPSS*, John Wiley & Sons, New York, NY 1974.

[SHE84] Sheppard, Sallie, Udo Pooch and C. Dennis Pegden, eds., *1984 Winter Simulation Conference Proceedings*, Society for Computer Simulation, La Jolla, Calif. 1984.

[SIG84] *1984 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM Order #488840, ACM, Baltimore, Md.

[SUR83] Suri, Rajan "Infinitesimal Perturbation Analysis of Discrete Event Dynamic Systems: A General Theory", *Proceedings of 22nd IEEE Conference on Decision and Control*. December 1983.

Manuscript received October 1985.

Mr. Zettel has a B.S. in Chemical Engineering in 1959 from a school he would just as soon not name. He also has an MBA (1975) from Canisius College. Mr. Zettel did his first simulation model in 1962, using GPSS to study the interaction of specification and aging constraints during propellant filling on the final quality of Polaris rocket motors while working for the Aerojet-General Corporation.

He has since done simulation studies for a number of companies using various simulation languages and techniques, including GPSS, GASP IV, SLAM II, and SIMAN. Currently a member of the research staff of the Ford Motor Company, Mr. Zettel's interests center on the simulation of manufacturing systems.

```
SCREEN #17
\ RANDOM NUMBER GENERATOR
\ FROM STARTING FORTH P265.
CREATE RND 0 , HERE RND !
: RANDOM RND @ 31421 * 6927 + DUP
  RND ! ;
: CHOOSE ( U1 --- U2)
  RANDOM U* SWAP DROP ;

: <ROT (N1 N2 N3...N3 N2 N1) \ BACKWARDS ROT
  ROT ROT ;
```

```
SCREEN #36
\ FREE MIN.NODE GET
VARIABLE FREE 0 FREE ! 3 CONSTANT MIN.NODE
: GET ( N --- ADDR) \ PUT THE ADDRESS OF THE NEXT AVAILABLE
\ NODE OF SIZE N ON THE STACK.
 DUP FREE DUP @  \ FIND A FREE NODE BIG ENOUGH.
 BEGIN DUP
       IF DUP 2+ @ 4 PICK <
       ELSE DUP THEN
 WHILE SWAP DROP DUP @ REPEAT ?DUP
 IF DUP 2+ @ 4 PICK - DUP MIN.NODE <  \ SALVAGE REST OF NODE?
    IF DROP DUP @ ROT ! SWAP DROP
    ELSE OVER 5 ROLL 2* + DUP 5 ROLL ! 3 PICK @ OVER ! 2+ !
    THEN
 ELSE DROP HERE SWAP 2* ALLOT  \ NO NODE; MAKE ONE.
 THEN DUP <ROT 2+ ! ;
```

```
SCREEN #37
\ N! N@ 2+! GIVE
: N! ( N1...NN ADDR N ---)
\ STORE N1 TO NN IN CONSECUTIVE CELLS STARTING AT ADDR.
  2* 0 DO SWAP OVER I + ! 2 +LOOP DROP ;
: N@ ( ADDR N --- N1...NN) \ FETCH N CONSECUTIVE VALUES STARTING
     \ AT ADDR + 2N-2 & LEAVE THEM ON THE STACK.
  1- 2* 0 SWAP DO DUP I + @ SWAP -2 +LOOP DROP ;

: 2+! ( D1 ADDR ---) \ ADD D1 TO THE DOUBLE NUMBER AT ADDR.
 DUP >R 2@ D+ R> 2! ;

: GIVE ( ADDR ---)
\ ADD NODE AT ADDR TO FREE LIST.
 FREE @ OVER ! FREE ! ;



SCREEN #38
\ PUSH POP
: PUSH ( N1...NN ADDR ---) \ PUSH N1 TO NN ONTO THE STACK
       \ POINTED TO BY ADDR.  NN IS THE NODE SIZE IN CELLS.
 OVER GET OVER @ SWAP DUP 4 ROLL !
 3 PICK N! ;

: POP ( ADDR --- N1...NN) \ POP STACK POINTED TO BY ADDR,
\ LEAVING VALUES ON STACK & NODE ON FREE LIST.
  DUP @ DUP NOT ABORT" EMPTY USER STACK."
  DUP @ ROT ! DUP GIVE 2+ DUP @ 1- N@ ;
: ME ( --- ADDR) \ PUT THE CFA OF THE WORD BEING
\ COMPILED ON THE STACK.
  LATEST PFA CFA [COMPILE] LITERAL ; IMMEDIATE




SCREEN #39
\ 2PICK CALENDAR SCHEDULE
: 2PICK ( N--- D1) \ RETURN THE CONTENTS OF THE NTH AND N+1ST
\ STACK VALUES, NOT COUNTING N.
  1+ DUP 1+ PICK SWAP PICK ;

VARIABLE CALENDAR 0 CALENDAR !
: SCHEDULE ( N1...NN ADDR DTIME M ---) \ PUT THE EVENT WITH
\ COMPILATION ADDRESS ADDR ON THE EVENT CALENDAR.

 CALENDAR DUP @
 BEGIN DUP IF 4 2PICK 3 PICK 4 + 2@ D>
           ELSE DUP THEN
 WHILE SWAP DROP DUP @ REPEAT
 DROP PUSH ;


```

```
SCREEN #40
\ CLOCK NEXT.EVENT SIMULATE END.RUN
2VARIABLE CLOCK
: NOW ( --- D1) \ D1 IS THE CURRENT SIMULATION CLOCK TIME.
 CLOCK 2@ ;
: NEXT.EVENT ( ---) \ TAKE THE NEXT EVENT OFF THE CALENDAR,
\ UPDATE THE SIMULATION CLOCK, AND EXECUTE THE EVENT.
 CALENDAR @ NOT ABORT" NO NEW EVENT"
 CALENDAR POP DROP CLOCK 2! EXECUTE ;

: SIMULATE ( ---)
 BEGIN NEXT.EVENT ?TERMINAL UNTIL ;

: END.RUN ( ---)
 ABORT" END OF SIMULATION." ;


SCREEN #88
\ TRUE FALSE WAITING BARBER BALKS SERVED FINISH.HAIRCUT
1 CONSTANT TRUE 0 CONSTANT FALSE
VARIABLE WAITING VARIABLE BARBER
VARIABLE BALKS VARIABLE SERVED
VARIABLE ARRIVAL.MIN VARIABLE ARRIVAL.SPREAD
VARIABLE HAIRCUT.MIN VARIABLE HAIRCUT.SPREAD
: NOW. ( ---) \ PRINT THE CURRENT SIMULATION TIME.
 NOW 3600 U\MOD
 . ." HOURS " 60 \MOD . ." MINUTES " . ." SECONDS. " ;
: FINISH.HAIRCUT ( ---)
 1 SERVED +! WAITING @
 IF -1 WAITING +! ME HAIRCUT.MIN @ HAIRCUT.SPREAD @ CHOOSE
   + S->D NOW D+ 5 SCHEDULE
 ELSE FALSE BARBER !
 THEN ;

SCREEN #89
\ CUSTOMER.ARRIVES SETUP.BARBERSHOP
: CUSTOMER.ARRIVES ( ---)
 ME ARRIVAL.MIN @ ARRIVAL.SPREAD @ CHOOSE +
 S->D NOW D+ 5 SCHEDULE BARBER @
 IF WAITING @ 4 <
    IF 1 WAITING +! ELSE 1 BALKS +! THEN
 ELSE TRUE BARBER ! ['] FINISH.HAIRCUT CFA HAIRCUT.MIN @
      HAIRCUT.SPREAD @ CHOOSE + S->D NOW D+ 5 SCHEDULE
 THEN ;
: SETUP.BARBERSHOP ( ---) \ INITIALIZE THE SIMULATION.
 0 WAITING ! FALSE BARBER ! 0 BALKS ! 0 SERVED !
 0. CLOCK 2! CUSTOMER.ARRIVES ;
```

```
SCREEN #96
\ BARBERSHOP REPORT
: REPORT ( ---)
 CR ." AFTER " NOW. CR SERVED @ .
 ." CUSTOMERS HAVE BEEN SERVED," CR
 WAITNG @ . ." ARE WAINTING AND" CR BALKS @ .
 ." HAVE BALKED." CR ;
```

```
SCREEN #97
\ DEMO LOAD SCREEN FOR BARBER SHOP SIMULATION.
: ELAPSED.TIME ( D1 D2 ---) 2SWAP D- DROP 100 60 */ S->D
 <# # # 46 HOLD #S #> TYPE ." SECONDS " ;
 HERE RDTIM 17 DUP . LOAD 36 40 THRU
 CR ." SYSTEM COMPILATION" CR ROT HERE SWAP - . ." BYTES "
 RDTIM ELAPSED.TIME HERE RDTIM
 88 89 THRU 96 DUP . LOAD
 CR ." MODEL COMPILATION" CR ROT HERE SWAP - . ." BYTES " RDTIM
 ELAPSED.TIME RDTIM
 : END.SIM ( D1 ---) REPORT RDTIM ELAPSED.TIME ." EXECUTION." CR
 ABORT" END OF SIMULATION" ;
 720 ARRIVAL.MIN ! 720 ARRIVAL.SPREAD !
 720 HAIRCUT.MIN ! 480 HAIRCUT.SPREAD !
 SETUP.BARBERSHOP
 ' END.SIM CFA 28800. 5 SCHEDULE SIMULATE
```