Two Contributions to the Techniques of Queuing Problems

By C. Strachey

Programs which simulate queuing problems are often disappointingly slow because it is necessary to investigate a very large number of small time intervals. Moreover, as the amount of information required for each interval is generally small, it is often necessary for reasons of economy to pack the information for several intervals into a single word. The process of packing and unpacking this information is tiresome and time consuming, and it would obviously be very advantageous to be able to treat a complete word of several packed intervals simultaneously and in parallel, without unpacking it. As queuing processes are essentially sequential, it might seem at first sight that this was an impossible aim; it has, however, proved possible to do this in certain simple cases, and the purpose of this note is to describe the techniques employed in the hope that they will have further uses.

The basic idea is that each binary digit position represents one interval of time (this presupposes the use of a fixed-point binary machine). Information of different types is kept in different words, and operations concerning a single time interval are carried out by bit-wise logical operations on all the bits of the word in parallel. The least-significant end of the word represents the earliest interval in time, so that the operation of "carrying" in the arithmetic unit can be used to transfer information from earlier intervals to later ones.

Random Numbers

It is often necessary to simulate events which occur at random but at a fixed average rate. For our representation we therefore need a random number such that each digit individually has a given probability f of being a 1.

We can construct these numbers without considering each digit separately as follows:

$$y_0 = I.$$

For $n = 0, 1, 2, 3, \ldots$

$$x_{n+1} = y_n \;\&\; R_n$$
$$y_{n+1} = y_n \;\&\; \overline{R_n} = y_n - x_{n+1}.$$

Here I represents a word which is all 1's, and R_n represents a random number of the ordinary sort with an equal probability of 0's and 1's; a bar is used for complementation (interchanging 0's and 1's) and & is used for collation (the operation which puts 1's in the result only where there are 1's in both operands).

It is clear that x_{n+1} and y_{n+1} have no 1's in common and their sum is y_n; furthermore, each of them has on the average one half of the number of 1's of y_n. Thus $x_1, x_2, \ldots$ form a sequence of disjoint random numbers such that each digit of x_n has a probability of 2^{-n} of being a 1. We can now form our required random number by adding together those x_n which correspond to the 1's in the binary representation of the fraction f.

This process can be quite easily programmed in the form of a loop which terminates either when the y_n becomes zero or when the digits of f run out. For a computer of normal word length the cycle will be traversed about six or seven times (normally terminating when $y_n = 0$), and will be very much faster than a digit-by-digit process.

Simple Queues

The situation considered is like trying to load irregularly arriving objects on to a conveyor belt which is already partly full. There are certain empty slots available on the belt and if there is a queue waiting when a slot arrives it will be filled. If there is no queue waiting and no object arrives at the same time as the slot it will not be filled, and any object arriving subsequently will have to wait until the next slot. In the first instance, we shall assume that there is no queue at the start of the time we are considering; we shall see later what modifications are necessary to deal with the case in which there is.

Let the slots available on the belt be represented by the word S_0, and the incoming stream of objects by N_0. We first deal with digits which are in the same position in S_0 and N_0, since we know at once that there must be an output in these positions (we refer to objects which are successfully placed into slots on the belt as "output" and will use a letter T for them). The existence or otherwise of a queue at this moment is immaterial, since the objects are considered to be indistinguishable. We therefore put

$$T_0 = S_0 \;\&\; N_0$$
$$S_1 = S_0 - T_0$$
$$N_1 = N_0 - T_0.$$

Thus T_0 is the contribution to the output, and S_1 and N_1 the remaining parts of the slots and the input respectively.

We now form $D_1 = S_1 - N_1$. In order to see what happens here, let us divide S_1 into sections of one or more digits. Each section starts at a 1 of S_1 and continues up to but not including the next 1 to the right; N is divided into the same sections. Thus, while each section of S consists of a 1, possibly followed by some 0's, the corresponding section of N starts with a 0 (since we have already removed the 1's common to S and N)

and may be followed by either 0's or 1's. If the section of N is all 0's the corresponding section of D will be identical with that of S. If, however, the section of N contains one or more 1's, the section of D will have the following properties:

(*a*) The most significant digit will be a 0, corresponding to the 1 in the section of S.

(*b*) There will be one and only one 1 in the same position in D and N, and this will be the least significant 1 in N.

Thus we can now form

$$D_1 = S_1 - N_1$$
$$P_1 = N_1 \ \& \ D_1$$
$$T_1 = S_1 \ \& \ \overline{D_1}$$

where P_1 represents the objects removed from N and inserted at a later time T_1 into the slots S. We can now form the remaining

$$N_2 = N_1 - P_1$$
$$S_2 = S_1 - T_1.$$

The entire process (starting with the formation of D_2) must now be repeated until there are no further contributions to the output, i.e. until $T_r = 0$. The final output is the sum of all the T's.

An example may help to make this clearer (see Table 1).

There remains the problem of initial and final queues, that is to say the carry-over from previous periods and into subsequent ones. An initial queue can always be considered as a source of digits at the least significant end of N. The simplest way of dealing with it is probably to use S–N–1 for D, in place of S–N, in any step in which the initial queue is not empty; at the same time the initial queue should be reduced by 1. A final queue will only originate when there are 1's in N more significant than any surviving 1's in S. This will mean that there are objects arriving for which no slots can be found. It will be demonstrated by the fact that D will appear negative. In these circumstances, 1 should be added to the final queue because the ordinary cycle will reduce the number of 1's in N by 1 without any corresponding output. When the cycle finishes, any initial queue still remaining should be added to the final queue and, if N is not zero, its sideways total (i.e. the number of 1's in it), should also be added to the final queue.

The maximum length of the queue at any time can be determined by counting the number of times the cycle is traversed and adding the number, if any, remaining in the initial queue at the end of the process. It is also useful to be able to find the total time spent by objects

Table 1 — Example

	S_0	1	0	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0	1	1	1
	N_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	0	0	1	1	1	0
S_0 & N_0 =	T_0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0
$S_0 - T_0 =$	S_1	1	0	1	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1
$N_0 - T_0 =$	N_1	0	1	0	0	0	0	1	1	0	1	0	0	0	1	0	0	1	0	0	0
$S_1 - N_1 =$	D_1	0	1	1	1	0	0	0	0	1	1	1	0	1	1	0	0	1	0	0	1
S_1 & $\bar{D}_1 =$	T_1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0
N_1 & $D_1 =$	P_1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
$S_1 - T_1 =$	S_2	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
$N_1 - P_1 =$	N_2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
$S_2 - N_2 =$	D_2	0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	1
	T_2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P_2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	S_3	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
	N_3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	D_3	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	1
	T_3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P_3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	S_4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
	N_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
								$D_4 = S_4$				$T_4 = 0$									
Output	ΣT	1	0	1	1	0	1	0	0	0	0	0	1	1	0	0	1	0	1	1	0

in the queue, as this makes it possible to find the average number of objects in the queue at any moment, and the average time in the queue spent by each object. This can be done by forming the sideways total of the quantities $T_r - P_r$ and making suitable provision for the waiting time of objects transferred directly from the initial queue to the final queue and for those still left in N at the end of the process. Another quantity which might be wanted is the maximum time spent in the queue by any one object; unfortunately, it seems impossible to obtain this number using this sort of technique, as the identity of the objects is not preserved.

The description of these processes has deliberately been left somewhat general, and detailed algorithms have not been given, because the logical orders available vary a good deal from computer to computer, and the exact details of what is available will have a very considerable effect on the actual program. Examples of the Pegasus programs for the two processes are given in the Appendix.

Appendix: Examples of Pegasus programs

Note.—The sign bit is not used.

Random Numbers

```
X2   Output
X3   f   Probability for each digit to be 1
X4   y_n
X5   R_n   Ordinary random number (sign bit = 0)

f     3 00
R_0   5 00
1     4 42 }
32    4 06 } y_0 = 1's
0     2 00

→5    5 20 }
 6    7 06 } X5 = X7 = R_n
 7    5 00 }
 4    7 05   y_n & R_n = x_{n+1}
 7    4 03   y_n - x_{n+1} = y_{n+1}
 1    3 52 }
      3 62 } Test next digit of f
 7    2 01   Add x_{n+1} if necessary
 →    3 60
      4 61
      EXIT←
```

Note.—The method of generating random numbers used above appears to be novel; it is certainly fast, convenient, and satisfactory.

Simple Queues

```
X2   Q   Initial queue
X3   Q*  Final queue
X4   N
X5   S

0     3 00   Q* = 0
S_0   5 10   Store S_0
4     5 05   T_0 = N_0 & S_0
5     4 03   N_1
S_0   5 04   S_1

→5    7 00 }
 4    7 03 } X7 = D
      2 60
 1    7 43   D - 1 } Deal with
 1    2 43   Q - 1 } initial queue
 →    7 62
 1    3 41   Final queue
→5    6 00
 7    5 05   S & D
 5    6 03   S - S & D = S & D̄ = T
 4    7 05   N & D = P
 7    4 03   N - N & D = N & D̄
      4 61   Jump if N ≠ 0
      6 61   Jump if T ≠ 0
S_0   5 04   X5 = ΣT = final output
2     3 01   Final queue
      EXIT
```

More sophisticated methods of testing can be used to avoid the last repetition of the cycle, which is superfluous.