[1]$\overline{1}$

27

Concurrent Reading and Writing of Clocks

Leslie Lamport

April 1, 1988, revised November 20, 1990

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in l984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Concurrent Reading and Writing of Clocks

Leslie Lamport

April 1, 1988
Revised November 20, 1990

Author’s Abstract

As an exercise in synchronization without mutual exclusion, algorithms are developed to implement both a monotonic and a cyclic multiple-word clock that is updated by one process and read by one or more other processes.

Capsule Review

It is convenient for an operating system to maintain the system clock in shared memory, so it can be read directly by user processes, without a system call. But doing this is tricky if the clock has more than one word of precision, because the system may update the clock while a user process is partway through reading it. This paper presents a simple algorithm for maintaining the clock in shared memory that requires no locking or retries. Theorists will find the algorithm and its correctness proof interesting, while practitioners will find the algorithm useful and easy to implement.

Tim Mann

Contents

1 Introduction

In an asynchronous multiprocess system, consider a clock that is updated by one process and read by one or more other processes. The clock is represented as a sequence of digits, where reading or writing each digit is a separate operation. We seek an algorithm to guarantee that a process reads a correct clock value, even if the read is performed while the clock is being updated. A read that occurs while the clock is being changed from 11:57 to 12:04 is allowed to return any value between 11:57 and 12:04 (inclusive). However, it is not allowed to return values such as 11:04, 12:07, or 12:57, which could be obtained if no attempt were made to synchronize the reader and writer. The relevance of this problem to the implementation of a multiword clock in an operating system should be obvious.

It is widely believed that this problem can be solved only by "locking"—that is, by using a mutual exclusion protocol to prevent the reader and writer from concurrently accessing the clock. This belief is wrong. Using the results of [1], I will derive a solution in which processes never wait. Such a solution is likely to be more efficient than one that uses locking. It is the goal of this paper not only to present a potentially useful algorithm, but also to remind readers that one can sometimes avoid the need for mutual exclusion by using the techniques of [1].

It is customary to assume that reading or writing a single digit is an atomic operation. The algorithms presented here and the theorems of [1] on which they are based are valid under the weaker assumption that each digit is, in the terminology of [2], a regular register. (The version of [1] submitted for publication assumed only regular registers, but the editor was afraid that the concept of nonatomic operations on individual digits might be considered heretical and insisted that it be removed from the paper.) However, readers who wish to ignore the subtle distinction between atomic and regular digits can simply think "atomic" when they read "regular".

There are two different kinds of clocks—*monotonic* clocks that never decrease and *cyclic* clocks that cycle through a bounded set of values. Monotonic clocks are commonly used in operating systems to encode the date and time, while cyclic clocks that display the time of day are encountered in everyday life. I will first present an algorithm to implement a monotonic clock and then indicate how it can be extended to implement a cyclic clock.

2 Notation and Theorems

To begin, let us recall the notations and results of [1]. Write operations to a data item v are assumed to be sequential; the sequence of values written to v is denoted by $v^{[0]}$, $v^{[1]}$, $\ldots$, where the write of $v^{[i]}$ precedes the write of $v^{[i+1]}$. The 0^{th} write, which initializes the value, is assumed to precede all reads. A read of v is said to *see version* $v^{[i]}$ if the read ended after the write of $v^{[i]}$ was begun and began before the write of $v^{[i+1]}$ ended. A read that is concurrent with a write will see more than one version.

In a somewhat illogical but useful notation, for natural numbers k and l with $k \leq l$, let $v^{[k,l]}$ denote both the value obtained by a read and the assertion that the read saw versions $v^{[k]}$, $v^{[k+1]}$, $\ldots$, $v^{[l]}$ and no other versions. Thus, if a read obtains the value $v^{[k,l]}$, then either $k = l$ and the read overlapped no writes, or $k < l$ and the read overlapped the writes of $v^{[k+1]}$, $\ldots$, $v^{[l]}$.

A data item v is said to be *regular* if the value $v^{[k,l]}$ obtained by a read equals $v^{[i]}$ for some i with $k \leq i \leq l$. This is a somewhat weaker condition than requiring that reads and writes of v be atomic.

An m-digit data item v is a sequence $v_1 \cdots v_m$ of digits, where the left-most digit v_1 is the most significant. The ranges of values of the v_i need not all be the same; mixed-radix representations are allowed. A read or write of v consists of a single read or write of each digit, in any order. Thus, $v^{[i]} = v_1^{[i]} \cdots v_m^{[i]}$. A read or write of v is said to be from *left to right* if the digits are accessed in the order v_1, v_2, $\ldots$, v_m; it is said to be from *right to left* if the digits are accessed in the opposite order. The following theorems and lemma are proved in [1]. (Theorem 2 will not be used here, but is included for completeness.)

Theorem 1 *If $v = v_1 \cdots v_m$ is always written from right to left, then a read from left to right obtains a sequence of values $v_1^{[k_1,l_1]}$, $\ldots$, $v_m^{[k_m,l_m]}$ with $k_1 \leq l_1 \leq k_2 \leq \ldots \leq k_m \leq l_m$.*

Lemma *Let $v = v_1 \cdots v_m$ and assume that $v^{[0]} \leq v^{[1]} \leq \cdots$.*

(a) *If $i_1 \leq \cdots \leq i_m \leq i$ then $v_1^{[i_1]} \cdots v_m^{[i_m]} \leq v^{[i]}$.*

(b) *If $i_1 \geq \cdots \geq i_m \geq i$ then $v_1^{[i_1]} \cdots v_m^{[i_m]} \geq v^{[i]}$.*

Theorem 2 *Let $v = v_1 \cdots v_m$ and assume that $v^{[0]} \leq v^{[1]} \leq \cdots$ and the digits v_i are regular.*

(a) *If v is always written from right to left, then a read from left to right obtains a value $v^{[k,l]} \leq v^{[l]}$.*

(b) *If v is always written from left to right, then a read from right to left obtains a value $v^{[k,l]} \geq v^{[k]}$.*

The reader who enjoys puzzles may wish to pause here and attempt his own implementation of a monotonic clock using regular digits.

3 A Monotonic Clock

A monotonic clock is a data item $c = c_1 \cdots c_m$ such that $c^{[0]} \leq c^{[1]} \leq c^{[2]} \leq \cdots$. The correctness condition for such a clock asserts that if a read obtains the value $c^{[k,l]}$, then $c^{[k]} \leq c^{[k,l]} \leq c^{[l]}$. The individual digits c_i are assumed to be regular.

To implement a monotonic clock c, two copies $c1$ and $c2$ of the clock are maintained. The writer updates c by first writing $c2$ from left to right and then writing $c1$ from right to left. The reader first reads $c1$ from left to right and then reads $c2$ from right to left. In the following analysis, we deduce what value the reader should return.

Let $r1$ and $r2$ denote the values of $c1$ and $c2$ read by the reader. By Theorem 1 applied to the $2m$-digit data item $c1_1 \cdots c1_m c2_m \cdots c2_1$,

$$r1 = c1_1^{[k1_1, l1_1]} \cdots c1_m^{[k1_m, l1_m]} \tag{1}$$

$$r2 = c2_1^{[k2_1, l2_1]} \cdots c2_m^{[k2_m, l2_m]} \tag{2}$$

with

$$\begin{aligned} k1_1 \leq l1_1 \leq k1_2 \leq \cdots \leq k1_m \leq l1_m \\ \leq k2_m \leq l2_m \leq k2_{m-1} \leq \cdots \leq k2_1 \leq l2_1 \end{aligned} \tag{3}$$

The regularity assumption and (1)–(3) imply the existence of integers i_q and j_q such that

$$r1 = c1_1^{[i_1]} \cdots c1_m^{[i_m]} \tag{4}$$

$$r2 = c2_1^{[j_1]} \cdots c2_m^{[j_m]} \tag{5}$$

and

$$k1_1 \leq i_1 \leq \cdots \leq i_m \leq j_m \leq \cdots \leq j_1 \leq l2_1 \tag{6}$$

The definition of $c^{[k,l]}$ implies that $k \leq k1_1$ and $l2_1 \leq l$, so a correct read is allowed to return any value in the interval $[c^{[i_1]}, c^{[j_1]}]$.

Applying part (a) of the lemma to $c1$ with i_m substituted for i, and part (b) to $c2$ with j_m substituted for i, using (4) and (5), we get $r1 \leq c1^{[i_m]}$ and $c2^{[j_m]} \leq r2$. The monotonicity assumption and (6) then imply

$$r1 \leq c1^{[i_m]} \leq c2^{[j_m]} \leq r2 \tag{7}$$

Monotonicity, (6), and the assumption that $c1$ and $c2$ are just two copies of c imply

$$c^{[i_1]} = c1^{[i_1]} \leq c1^{[i_m]} \leq c2^{[j_m]} \leq c2^{[j_1]} = c^{[j_1]} \tag{8}$$

If $r1 = r2$, then (7) and (8) imply that the read can return the value $r1$, since it can return any value in the interval $[c^{[i_1]}, c^{[j_1]}]$. Because (7) implies that $r1 \leq r2$, we need now consider only the case of $r1 < r2$.

For $1 \leq q \leq m$, let $v_{1 \cdots q}$ denote $v_1 \cdots v_q$, and let $v_{1 \cdots 0} = 0$. Define $v\!\downarrow_q$ and $v\!\uparrow_q$ to be the smallest and largest m-digit values w such that $w_{1 \cdots q} = v_{1 \cdots q}$. Thus, $v_{1 \cdots q}$ consists of the left-most q digits of v, and, if the v_i are decimal digits, then $v\!\downarrow_q$ and $v\!\uparrow_q$ are the values obtained by replacing the right-most $m - q$ digits of v with 0's or 9's, respectively. In general, we have

$$v\!\downarrow_q \leq v \leq v\!\uparrow_q \tag{9}$$

for $1 \leq q \leq m$.

Since we are assuming that $r1 < r2$, there exists a unique p, with $0 \leq p < m$, such that

$$r1_{1 \cdots p} \quad = \quad r2_{1 \cdots p} \tag{10}$$

$$r1_{p+1} \quad < \quad r2_{p+1} \tag{11}$$

From (11) we have

$$r1\!\uparrow_{p+1} < r2\!\downarrow_{p+1} \tag{12}$$

Applying part (a) of the lemma to $c1_{1 \cdots p}$ with i_{p+1} substituted for i and p substituted for m, and part (b) to $c2_{1 \cdots p}$ with j_{p+1} substituted for i and p substituted for m, we obtain

$$r1_{1 \cdots p} \quad \leq \quad c1_{1 \cdots p}^{[i_{p+1}]} \tag{13}$$

$$r2_{1 \cdots p} \quad \geq \quad c2_{1 \cdots p}^{[j_{p+1}]} \tag{14}$$

```
Reader
$r1 := \overrightarrow{c1}$;
$r2 := \overleftarrow{c2}$;
if $r1 = r2$
   then return $r1$
   else  $p := \max\{i : r1_{1\cdots i} = r2_{1\cdots i}\}$;
         return any value in $[r1{\uparrow_{p+1}}, r2{\downarrow_{p+1}}]$
fi
```

```
Writer
$\overrightarrow{c2}$ := any value $\geq c2$;
$\overleftarrow{c1} := c2$
```

Figure 1: The Monotonic-Clock Algorithm

Since $i_{p+1} \leq j_{p+1}$ by (6), monotonicity implies that $c1^{[i_{p+1}]} \leq c2^{[j_{p+1}]}$, so $c1^{[i_{p+1}]}_{1\cdots p} \leq c2^{[j_{p+1}]}_{1\cdots p}$. Hence, (10), (13), and (14) imply

$$r1_{1\cdots p} = c1^{[i_{p+1}]}_{1\cdots p} = c2^{[j_{p+1}]}_{1\cdots p} = r2_{1\cdots p}$$

By (4) and (5), this implies

$$r1_{1\cdots p+1} = c1^{[i_{p+1}]}_{1\cdots p+1} \tag{15}$$

$$c2^{[j_{p+1}]}_{1\cdots p+1} = r2_{1\cdots p+1} \tag{16}$$

Combining (9), (15), (16), and (12) yields

$$c1^{[i_{p+1}]} \leq c1^{[i_{p+1}]}{\uparrow_{p+1}} = r1{\uparrow_{p+1}} < r2{\downarrow_{p+1}} = c2^{[j_{p+1}]}{\downarrow_{p+1}} \leq c2^{[j_{p+1}]}$$

Hence, the reader can return any value in the interval $[r1{\uparrow_{p+1}},\ r2{\downarrow_{p+1}}]$.

The algorithm that has just been derived is shown in Figure 1, where an arrow over a variable name means that the corresponding read or write is performed left-to-right or right-to-left, as indicated by the arrow's direction. Note that it does not matter how the writer reads $c2$, since it is the only process that changes $c2$.

As a final optimization, observe that the reader reads $c1_m$ immediately before reading $c2_m$, while the writer writes $c2_m$ immediately before writing $c1_m$. The algorithm remains correct if the two reads or writes are performed as a single operation. Hence, the two digits $c1_m$ and $c2_m$ can be implemented by the same digit, which is read and written just once. In the most common application, $m = 2$ and reading or writing the clock requires only three single-digit reads or writes. A version of the algorithm for a two-digit clock is shown in Figure 2. A clock value is a pair (l, r) where l is the left digit and r the right digit, and 0 is assumed to be the smallest possible right digit.

```
Reader
$v1$ := $l1$;
$w$  := $r$;
$v2$ := $l2$;
if $v1 = v2$ then return ($v2$, $w$)
              else  return ($v2$, 0)
fi
```

```
Writer
($l'$, $r'$) := any value $\geq$ ($l2$, $r$);
$l2$ := $l'$;
$r$  := $r'$;
$l1$ := $l'$
```

Figure 2: A Two-Digit Monotonic-Clock Algorithm

4 A Cyclic Clock

A cyclic clock c is a data item that can assume any sequence of values. A write that decreases the value of c is said to *cycle c*. The cycling of c is interpreted to mean that c has “passed midnight”. For notational convenience, assume that 0 is the smallest value c can assume.

We can convert a cyclic clock c to a monotonic clock $\overline{\overline{c}}$ by adding a fictitious left-hand part that is incremented whenever the value of c is decreased. The correctness condition for c is that the value returned by a read is the right-hand part of a correct value for a read of $\overline{\overline{c}}$. Thus, if c is cycled during the read, then the read may return the value 0. If c is cycled twice during the read, then the read may return any value.

To construct an algorithm for implementing an m-digit cyclic clock $c = c_1 \cdots c_m$, we first augment c to an $m + 1$-digit cyclic clock c by adding an extra left-most binary digit c_0, so $c = c_0 \cdots c_m$. The left-most bit c_0 of c is thus incremented (complemented) whenever the clock c is cycled. The digits $c_0, \ldots, c_m$ are assumed to be regular.

The reads and writes of c are performed as in the monotonic-clock algorithm, so the writer begins by writing $c2_0$ and ends by writing $c1_0$, while the reader reads $c1_0$ first and $c2_0$ last. If the reader finds $c1_0 \neq c2_0$, then the read overlapped a write that cycled c, so the read can return the value 0. If the reader finds $c1_0 = c2_0$, then either c was not cycled during the read or else it was cycled two or more times. In the first case, the monotonic-clock algorithm returns a correct value because it returns the value it would have obtained had it seen the entire fictitious clock $\overline{\overline{c}}$; in the second case, the read is permitted to return any value. Hence, in either case, the (right-most m digits of the) value obtained by the monotonic-clock algorithm is correct.

The reader should be suspicious of this kind of informal argument because it often leads to errors. However, since I know of no practical application of the cyclic-clock algorithm, I will leave its precise statement and rigorous correctness proof to the reader.

Acknowledgments

Tim Mann discovered an error in my original correctness proof of the monotonic clock algorithm and suggested several improvements to the presentation.

References

[1] Leslie Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, November 1977.

[2] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1:77–101, 1986.