Fair Mutual Exclusion with Unfair P and V Operations

Alain J. Martin
and
Jerry R. Burch

Computer Science
California Institute of Technology

5148:TR:84

Fair Mutual Exclusion with Unfair P and V Operations

Alain J. Martin
and
Jerry R. Burch

Computer Science Department
California Institute of Technology

5148:TR:84

July 1984

The research described in this paper was sponsored by
the Defense Advanced Research Projects Agency, ARPA Order No. 3771,
and monitored by the Office of Naval Research
under contract number N00014-79-C-0597

Fair Mutual Exclusion with Unfair P and V Operations

Alain J. Martin and Jerry R. Burch

Computer Science

California Institute of Technology

Pasadena, CA 91125

July 1984

Keywords: mutual exclusion, P and V operations, semaphores, split binary semaphore, critical section, fairness, individual starvation

Introduction

The semantics of P and V operations can be described by two axioms, called the boundedness and progress axioms. For a given pair of P and V operations on semaphore s, let $\mathbf{c}Ps$ and $\mathbf{c}Vs$ be the number of completed P and V operations on s respectively, and let $\mathbf{q}Ps$ be the number of currently suspended P operations.

Boundedness axiom:

$$0 \leq s \ \wedge \ \mathbf{c}Ps + s = \mathbf{c}Vs + s0 \tag{0}$$

Progress axiom:

$$\mathbf{q}Ps = 0 \ \vee \ s = 0 \tag{1}$$

where $s0$ is the initial value of s. (See [4].)

According to the progress axiom, when a V operation on s is completed while $\mathbf{q}Ps > 0$—and thus $s = 0$—the completion of $V(s)$ coincides with the completion of a $P(s)$—and thus s remains 0. This property will play a crucial role in the sequel.

Observe that if $\mathbf{q}Ps > 1$—i.e. more than one process is delayed at $P(s)$—there is no guarantee that a given delayed process will complete its $P(s)$ when a $V(s)$ is performed,. If $\mathbf{q}Ps > 1$ holds forever, a given process may remain delayed forever ("blocked"). P and V operations with this property are said to be *unfair*.

Consider the well-known solution to the mutual exclusion problem [1]. Each process that wants to access the critical section CS on a mutual exclusion basis performs "$P(m); CS; V(m)$" where $m = 1$ holds initially. With unfair $P(m)$ and $V(m)$, this solution to the mutual exclusion problem is unfair, i.e. when more than two processes share the CS, we cannot guarantee that a process delayed at $P(m)$ will enter the CS after a finite number of $V(m)$ actions.

For a finite but unknown number of processes, we want to construct a fair solution to the mutual exclusion problem, using only a fixed number of unfair P and V operations. In

[5], J.M. Morris has proposed a solution using three binary semaphores. Here we propose a simpler solution with two semaphores used together as a "split binary semaphore".

2. Split binary semaphore

The technique of the split binary semaphore was first used by C.A.R. Hoare in [3], and investigated by Edsger W. Dijkstra in [2].

A split binary semaphore is a set of semaphores—here, y and z—whose initial values sum to 1. Further, in each process using this set of semaphores, the sequence of P and V operations on this set is an alternation of P and V operations, starting with a P and ending with a V. Hence, if np is the number of processes whose last operation on the split binary semaphore is a P

$$np + y + z = 1 \tag{2}$$

(For a formal proof, see [4].) Furthermore, since $np \geq 0, y \geq 0, z \geq 0$

$$np \leq 1 \ \wedge \ y + z \leq 1. \tag{3}$$

And thus each program part between a P and a V is an atomic action.

In the solution proposed, the program of each process consists only of parts enclosed between a P operation and a V operation on the split binary semaphore. Such a program is called an SBS-program, and unless otherwise specified the term "action" refers to such an atomic action.

In addition to providing atomic actions, the split binary semaphore technique enforces a sequencing of the actions by the following "domino rule". If an action A starts with $P(y)$ and ends with $V(z)$, y is called the "opening" semaphore of A, and z the "closing" semaphore of A. Such an action will be denoted by $\langle y, z \rangle$.

Domino rule: In any computation evoked by an SBS-program, if action B immediately follows action A, the opening semaphore of B is the closing semaphore of A. □

Proof: Let $V(x)$ be the closing V action of A, and $P(?)$ be the opening P action of B.

By (2), $x = 0$ holds as a precondition of $V(x)$ and as a postcondition of $P(?)$ since $np = 1$. Hence, because of (0), $\mathbf{c}Vx - \mathbf{c}Px$ has the same value before $V(x)$ and after $P(?)$.

Since $\mathbf{c}Vx$ is increased by one, $\mathbf{c}Px$ is increased by one. □

Let X^* denote the type of sequences consisting of zero or more repetitions of the sequence X.

Definition: A sequence of type $\langle y, y \rangle^* \langle y, z \rangle \langle z, z \rangle^* \langle z, y \rangle$ is called a *session*. □

Corollary: Let p be an SBS-program using the split binary semaphore (y, z) with initial values $y = 1$, $z = 0$. And let T be the sequence of actions corresponding to a computation evoked by p. Then any prefix of T is a prefix of a sequence of type session*. □

V is a prefix of sequence U if V is a finite sequence and there exists a sequence W such that $U = VW$.

3. The Program

Each process requesting access to the critical section CS performs the *mutual exclusion program*:

$$
\begin{array}{l}
P(y);\\
[\,m = 0 \rightarrow \,|\,[n : \textbf{int};\\
\qquad\qquad n, m := 0, 1;\\
\qquad\qquad *[n \neq m \rightarrow n := m;\ V(y);\ P(y)];\ V(z)\\
\qquad\qquad]\,|\,\\
[\!]\, m > 0 \rightarrow m := m + 1;\ V(y)\\
];\\
P(z);\\
CS;\ m := m - 1;\\
[\,m > 0 \rightarrow V(z)\\
[\!]\, m = 0 \rightarrow V(y)\\
].
\end{array}
$$

Initially, $y, z, m = 1, 0, 0$. Variable m is shared by all processes. Each process uses a private variable n as indicated by the declaration of the local block $|\,[n : \textbf{int};\ldots]\,|$. $P0(y)$ and $P1(y)$ denote the textually first and second $P(y)$ action respectively.

3.1 General description of the solution

A process whose next P or V action is $P0(y)$ or that is delayed at $P0(y)$, is a "candidate". A process that has passed $P0(y)$ but not yet passed the following $P(z)$ has "checked-in". Hence, it is easy to verify that—with our definition of atomic actions—m is equal to the number of checked-in processes. In each session, the first process to check-in—called fp—plays a special role: it "helps" other candidates to check-in by repeatedly performing a $V(y)$ in the repetition $*[n \neq m \rightarrow \ldots]$. Process fp uses variable n to record the current value of m. When n remains equal to m between two steps of the repetition, implying that there are no more candidates delayed at $P0(y)$ at this point in the session, fp performs $V(z)$, thereby allowing a checked-in process to enter the CS. Each "checking-out" process, i.e. leaving the CS, allows a next process to enter the CS by the standard technique of "cascaded waking-up".

3.2 Correctness proof

• A simple observation of the structure of the program shows it is an SBS-program. Hence (2) and (3) hold, which guarantees mutual exclusion on the access to the critical section.

• It is easy to verify that $m > 0$ holds as a postcondition of any atomic action except the action $\langle z, y\rangle$. Hence inside each session, there is exactly one process that executes $\langle y, y\rangle$ with $m = 0$ as a precondition—namely, the first checked-in process fp.

• We prove that the solution is fair by proving that each process delayed at $P0(y)$ at some point of the computation completes $P0(y)$ during a session following that point. And each process completing $P0(y)$ during a session enters the CS during this session. The proof is a direct consequence of the following two theorems.

Consider the initial state where $y, z, m = 1, 0, 0$, and the set of candidates is not empty. And consider the class of *maximal* computations started in this state, i.e. the class of computations started in this state that terminate if and only if each process is either blocked at a P operation of the mutual exclusion program or is not engaged in the mutual exclusion program. Let T be the sequence of actions corresponding to such a computation. Obviously T is not the empty sequence.

Theorem 1: T has a prefix of type session. □

Theorem 2: $y = 1$ holds as a precondition of $\langle y, z \rangle$. □

i) Since $m = 0$ as postcondition of a session, when a session terminates $y, z, m = 1, 0, 0$ holds again. From theorem 1, any T is thus of type session*.
ii) Since $m = 0$ as postcondition of a session, any process that completes $P0(y)$ during a session enters the CS during this session.
iii) Assume that process p is delayed at $P0(y)$ at some point in T. From theorem 2 and the progress axiom, p completes $P0(y)$ during the session of T containing the next $\langle y, z \rangle$. (From theorem 1 such a $\langle y, z \rangle$ exists.) Hence p enters its CS during this session. Hence the solution is fair.

Proof of theorem 1: From the corollary of the domino rule, T fulfills one of the following conditions:
a) T has a prefix of type session,
b) T is of type $\langle y, y \rangle^k$, with $k > 0$,
c) T is of type $\langle y, y \rangle^* \langle y, z \rangle \langle z, z \rangle^*$,
d) T contains the infinite sequence $U = \langle y, y \rangle U$,
e) T contains the infinite sequence $W = \langle z, z \rangle W$.

We prove that b) through e) are excluded.
b) Assume T is of type $\langle y, y \rangle^k$ with $k > 0$. When the computation terminates, on the one hand fp is blocked at $P1(y)$ since T contains at least one $\langle y, y \rangle$ action from fp (due to the initialization $n, m := 0, 1$). On the other hand, $y = 1$ since no $V(z)$ action has been performed in T. Hence, $\mathbf{q}Py > 0 \ \wedge \ y > 0$, which contradicts the progress axiom.
c) Assume T is of type $\langle y, y \rangle^* \langle y, z \rangle \langle z, z \rangle^*$. When the computation terminates, on the one hand $z = 1$. On the other hand, there is a process delayed at $P(z)$ since $m > 0$ as a postcondition of any $V(z)$ and fp is not delayed at $P1(y)$ since it has completed $\langle y, z \rangle$. Hence, $\mathbf{q}Pz > 0 \ \wedge \ z > 0$, which contradicts the progress axiom.
d) Let us call $Y1$ a $\langle y, y \rangle$ action in which m is increased, and $Y2$ a $\langle y, y \rangle$ action in which m is not increased. Since i) m has an upper bound, ii) m is not decreased in a $\langle y, y \rangle$ action, the number of $Y1$ actions in U is finite.

Since i) $n = m$ holds initially, ii) $n \neq m$ holds as precondition of a $Y2$ action, iii) $n = m$ holds as a postcondition of a $Y2$ action, each $Y2$ action is preceded by a $Y1$ action. Hence the number of $Y2$ actions in U is finite. A contradiction.
e) The number of consecutive $\langle z, z \rangle$ actions in T is finite since $m \geq 0$, and each $\langle z, z \rangle$ action decreases m by one. □

Proof of theorem 2: Let L be the last $Y2$ action preceding an action $\langle y, z \rangle$; $n = m$ holds as a postcondition of L and as a precondition of $\langle y, z \rangle$. Hence no $Y1$ action is executed between those two actions.

Since L and $\langle y, z \rangle$ both belong to fp, $y = 1$ between the two actions. □

4. Conclusion

The above solution is one more illustration of the elegance and generality of the split binary semaphore technique. The domino rule provides a general proof technique for programs constructed with split binary semaphores.

Acknowledgements

Edsger W. Dijkstra and Martin Rem both suggested a simplification in the coding of the algorithm. Excellent comments from Wim Feijen, Netty van Gasteren, Jan van de Snepscheut, and Chuck Seitz led to a drastic improvement of the presentation.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

References

[1] E.W. Dijkstra, Cooperating Sequential Processes, in: F. Genuys (ed.),*Programming Languages* (Academic Press, 1968).

[2] E.W. Dijkstra, A tutorial on the split binary semaphore, in: M. Broy and G. Schmidt (eds.),*Theoretical Foundations of Programming Methodology*, (Reidel, 1982).

[3] C.A.R. Hoare, Monitors: an operating system structuring concept,*Comm. ACM* 12 (10) (1974) 548-557.

[4] A.J. Martin, An axiomatic Definition of Synchronization Primitives,*Acta Informatica* 16, (1981) 219-235.

[5] J.M. Morris, A starvation-free solution to the mutual exclusion problem,*Information Processing Letters* 8, (1979) 76-80.