LIMITATIONS OF DIJKSTRA'S SEMAPHORE PRIMITIVES AND PETRI NETS

S. Rao Kosaraju*
Department of Electrical Engineering
The Johns Hopkins University
Baltimore, Maryland 21218

ABSTRACT

Recently various attempts have been made to study the limitations of Dijkstra's Semaphore Primitives for the synchronization problem of cooperating sequential processes [3,4,6,8]. Patil [8] proves that the semaphores with the P and V primitives are not sufficiently powerful. He suggests a generalization of the P primitive. We prove that certain synchronization problems cannot be realized with the above generalization and even with arrays of semaphores. We also show that even the general Petri nets will not be able to handle some synchronization problems, contradicting a conjecture of Patil (P.28 [7]).

I. INTRODUCTION

In general, a synchronization problem deals with the problem of achieving a required co-ordination between a system of coordinating processes. A precise definition is still lacking. As a first step, we advocate that only autonomous systems of processes should be considered for this problem. If a system of processes responds to external processes, then we should include those processes also within the system.

Semaphores

A semaphore is an integer valued variable, whose initial value is non-negative. The only operations that could be performed on semaphores are P and V. Each P or V operation is considered indivisible. In the literature there seems to be a certain ambiguity over the semantics of programs written with P and V primitives.

First interpretation (P. 199 [1])

P(S): [S ← S-1; if S < 0 the caller places himself in the queue Q_S, enters the waiting state, and releases the processor]+

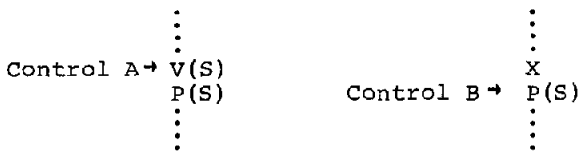
V(S): [S ← S+1; if S ≤ 0 remove some process from Q_S and add it to the work queue of the processors].

Second interpretation

P(S): [Can be executed when S > 0 which results in decrementing S by 1]

V(S): [S ← S+1]

The first interpretation seems to be convenient for implementation, and the second interpretation is easier for analysis. Another confusion is when we have a sequence of instructions $I_1, I_2, \ldots I_j, \ldots$, whether the execution of I_{j+1} starts immediately after the completion of I_j or there is an arbitrary delay in between. We assume arbitrary delays which seems to be the general understanding and also very natural in asynchronous operation. With this assumption, the theoretical distinction between the two interpretations disappears. For example, consider the following situation with S = 0.



B is waiting on the queue for S. As soon as A completes its V(S) operation, either B or A could execute a P(S) under the second interpretation; whereas under the first interpretation B is

*Supported by the U.S. Atomic Energy Commission under contract AT(11-1)-3288

+[X]: X is an indivisible operation.

awakened, since it is already on the queue, and immediately starts executing its P(S). Under the arbitrary delay assumption we could assume that B has not yet been placed on the queue, but instead has been delayed between X and P(S), and hence we can make A execute P(S) even for the first interpretation.

Petri nets

In a Petri net, there are a finite number of nodes, each node being a place or a transition. There are branches from places to transitions and from transitions to places. (We represent a process by a transition with one incoming branch and with one out-going branch). At any instant, each place holds an integer ≥ 0. The places which have branches to (from) a transition are the input (output) places of that transition.

At any instant, if every input place of a transition contains a number ≥ 1, then the transition could fire (but need not), which results in decrementing the stored number of each input place by 1 and in incrementing the stored number of each output place by 1. For a comprehensive discussion of Petri nets the reader is referred to [7].

II. A SYNCHRONIZATION PROBLEM

There are 2 Producers, P_1 and P_2, 2 Consumers, C_1 and C_2, and 2 Buffers, B_1 and B_2. When Producer P_i is activated, it produces an item and deposits it on top of the Buffer, $B_i (i = 1,2)$ then deactivates itself. When Consumer C_i is activated, it gets the bottom item of the Buffer, $B_i (i = 1,2)$ consumes it and then deactivates itself. Thus Buffers are resources, shared by producers and consumers, and each buffer is organized as a queue. To attempt to consume from an empty buffer is illegal. Initially each buffer is empty and producers are in an inactivate condition.

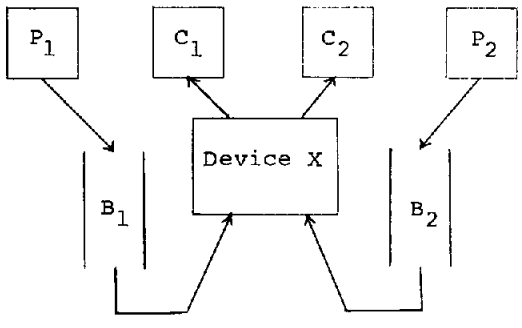


Figure 1: 2 Producer 2 Consumer Problem

As given in Figure 1, C_1 and C_2 communicate with their respective buffers through a common device X. We add the constraint, due to the limitations of X, that at any instant, C_1 and C_2 cannot be active simultaneously; C_1 has priority over C_2, i.e. if both C_1 and C_2 are both inactive and buffer B_1 is not empty, then C_2 cannot consume from B_2 (since C_1 can be activated) at that instant.

Some other similar synchronization problems are discussed in [5].

III. LIMITATIONS OF PETRI NETS

In this section we want to show that Petri nets cannot realize the 2 Producer 2 consumer problem.

Definition:

$(a_1,a_2,\ldots,a_n) \leq (b_1,b_2,\ldots,b_n)$ iff

$a_i \leq b_i$ for $i = 1,\ldots,n$.

Lemma 1:

Let A be any subset of N^n where $N = \{0,1,2,\ldots\}$. If no 2 distinct members of A are comparable under $\leq$, then A is a finite set.

Proof: Induction on n

n = 1: A could be $\emptyset$ (the empty set) or $\{i\}$ where $i \in N$. If $i_1, i_2 \in N$, then trivially, $i_1 \leq i_2$ or $i_2 \leq i_1$.

Let the Lemma hold for some $n \geq 1$.

Let $A \subseteq N^{n+1}$, be any set of pairwise incomparable elements under $\leq$. Let $(a_1,a_2,\ldots,a_{n+1}) \in A$. For $1 \leq i \leq n+1$ and $0 \leq j \leq a_i$, let

$$A_{i,j} = \{(c_1,c_2,\ldots,c_{i-1},c_{i+1},\ldots,c_{n+1}) \mid (c_1,c_2,\ldots,c_{i-1},j,c_{i+1},\ldots,c_{n+1}) \in A\}.$$

Each such $A_{i,j}$ must be a set of pairwise incomparable n-tuples. Hence, by inductive hypothesis, each $A_{i,j}$ is a finite set. In addition, for any $(b_1,b_2,\ldots,b_{n+1}) \in A$, $b_i \leq a_i$ for some i (otherwise $(a_1,a_2,\ldots,a_{n+1}) \leq (b_1,b_2,\ldots,b_{n+1})$).

Hence $\#A \leq \#\bigcup_{\substack{1 \leq i \leq n+1 \\ 0 \leq j < a_i}} A_{i,j}$.†

† #B = number of members of set B.

Any finite union of finite sets is a finite set. Hence A is a finite set. Q.E.D.

Thus the following Remark holds.

Remark 1: Let $X_1, X_2, X_3, \ldots$ be any infinite sequence of n-tuples of natural numbers. Then there exist i,j s.t. $i \neq j$ and $X_i \leq X_j$ or $X_j \leq X_i$.

Theorem 1: The 2 Producer 2 Consumer problem cannot be realized by Petri nets.

Proof: If not, let there exist such a Petri net F. Let there be n places in the net. n is a finite number. At any instant, when no transition is firing, the control state of the net is given by the n-tuple $(a_1, \ldots, a_n)$ where a_i gives the number stored in place i.

Lemma 2: At two different instants τ_1 and τ_2, if the corresponding control states X_1 and X_2 satisfy $X_1 \leq X_2$, then the sequence of transformations undergone from τ_1 can also be achieved from τ_2.

Proof: Trivial. At instant τ_1 if certain transitions fire resulting in control state X_1', then at instant τ_2 the same transitions can fire resulting in control state X_2' with $X_1' \leq X_2'$. Apply this inductively. Q.E.D.
Consider the sequence of activations shown in Figure 2.

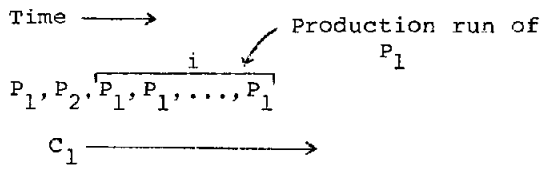


Figure 2: An activation Sequence

In this sequence, P_1 produces the first item and C_1 begins to consume this item. Since the synchronizer F must work for any relative speeds of Producers and Consumers, we may consider the case where C_1 takes so long in consuming the first item that P_2 may produce an item, followed by the production of i items by P_1, (Production run of P_1) for arbitrarily large i. This relation is indicated in the above diagram by the overlap of C_1 operation with the production sequence $P_2 P_1^i$.

Consider the control states after each production by P_1 in its production run. We recognize that many control state transformations may occur after any P_1 production and before the next P_1 production, but for this proof, we may pick any one of those control states as being representative. Thus for the production run P_1^i; we get a sequence of i control states. Let this sequence be $X_1, X_2, \ldots, X_i$. By Remark 1, for sufficiently large i, we can always find two distinct t_1 and t_2 s.t. $X_{t_1} \leq X_{t_2}$. Observe that the following activation sequence is valid.

$$P_1, P_2, \overbrace{P_1, P_1, \ldots, P_1}^{t_1}$$

$$C_1 \longrightarrow, \quad \underbrace{C_1, C_1, \ldots, C_1}_{t_1}, C_2$$

By Lemma 2,

$$P_1, P_2, \overbrace{P_1, P_1, \ldots, P_1}^{t_2}$$

$$C_1 \longrightarrow, \quad \overbrace{C_1, C_1, \ldots, C_1}^{t_1}, C_2$$

must be valid too. But the above is invalid since C_2 gets activated when B_1 is non-empty if $t_2 > t_1$ or C_1 gets activated when B_1 is empty if $t_2 < t_1$. Thus we get a contradiction. Q.E.D.

IV. LIMITATIONS OF SEMAPHORE PRIMITIVES

Patil [8] presents a synchronization problem, which he calls the Cigarette Smokers' Problem (denoted here by CSP), and proves that the necessary synchronization cannot be achieved with just semaphores and the P and V primitives. He suggests a generalization of P (denoted here by P_G) to include simultaneous operation over a finite number of semaphores; e.g. $P_G(S_1, S_2, S_3)$ can get executed when $S_1, S_2, S_3 \geq 1$ and the execution decreases each of S_1, S_2, S_3 by 1.

Subsequent to Patil's work, Parnas [6] observes that Dijkstra allows arrays of semaphores and using an array of semaphores he gives a realization of CSP. It is not clear what types of operations should be allowed on array index variables. However, at least the array index variable should not be allowed to take over any other function. Hence, we consider the following framework:

We allow arrays of semaphores, with the limitation that the total number of semaphores is finite and bounded. Thus, each array becomes finite and bounded; let the maximum size be q. We also allow a finite bounded number of array index variables: $Y_1, Y_2, Y_3, \ldots, Y_m$. We do not allow any Y_i taking a value greater than q. We place this restriction to make sure that the array index variables are used only as array indexes and not used in other disguised forms.

We allow any assignment statement of the form $Y_j \leftarrow f(Y_1, Y_2, \ldots, Y_m)$, where f is any function.

Theorem 2: The 2 Producer 2 Consumer problem cannot be realized with arrays of semaphores and P_G and V primitives.

Proof: If possible, let there exist such a synchronization control F. We could assume that each process (P_1, P_2, C_1 or C_2) is an instruction. Then the control net F is composed of the following types of instructions: P_G and V primitives, array index assignment statements, GOTO'S, and P_1, P_2, C_1 and C_2. Since there are no test instructions, the control can be decomposed into a set of sub-controls, where each sub-control is a sequence of non-GOTO instructions or a sequence of non-GOTO instructions followed by a GOTO a designated instruction in the sequence. Initially the point of control is at the start of each sub-control. In F, let there be n semaphores, $S_1, S_2, \ldots, S_n$, and m array index variables, $Y_1, Y_2, \ldots, Y_m$. At any instant when no instruction is being executed, the point of control within each sub-control will be in between 2 instructions. Note that there are only a finite number of possible control point combinations. Let there be δ sub-controls and for the i-th sub-control let there be b_i control points. Then the number of possible control point combinations $= b_1 b_2 \ldots b_\delta (= K)$. Denote these control point combinations by $e_1, e_2, \ldots, e_K$. At any instant when no instruction is being executed, the control state of the net F is given by (e, U, Z) where e is a control point combination, U is an m-tuple of values of the m array index variables and Z is an n-tuple of values of n semaphore variables. Let us define a relation $\leq$ between control states as follows:

$(e, U, Z) \leq (e', U', Z') \iff e = e',\ U = U',\ Z \leq Z'$.

Now the proof runs exactly as that of Theorem 1, starting from Lemma 2 (in the proof of Lemma 2 firing of a transition should be replaced by execution of an instruction). Q.E.D.

V. COMMENTS

In the CSP of Patil [8] there are 4 processes, an agent and three smokers, sitting at a table. One of the smokers has tobacco, another has cigarette papers, and the third has matches. Basically the agent throws 2 items on the table and communicates through semaphores, to the agents which 2 items have been placed. The Smoker with the third item should pick up those items, make a cigarette and smoke it. The agent's control net is pre-specified and the problem is to design the control for each smoker. The agent is described (using semaphores S,a,b,c) by:

R_a	R_b	R_c
r_a: P(S)	r_b: P(S)	r_c: P(S)
Throw Wrapper & Match	Throw Match & Tobacco	Throw Wrapper & Tobacco
V(b) V(c) goto r_a	V(a) V(c) goto r_b	V(a) V(b) goto r_c

Thus the agent throws 2 items and sets 2 semaphores to 1 (initially S = 1, a = b = c = 0).

Let us modify the agent as follows (using only semaphores S,a,b)

R_a	R_b	R_c
r_a: P(S)	r_b: P(S)	r_c: P(S)
Throw Wrapper & Match	Throw Match & Tobacco	Throw Wrapper & Tobacco
V(a) V(b) goto r_a	V(a) V(a) V(b) goto r_b	V(a) V(a) V(a) V(b) goto r_c

(Initially S = 1, a = b = 0).

This agent communicates (through semaphores a and b) to the Smokers enough information to indicate which 2 items have been thrown in. A major problem in the design of smokers will be information decoding to find which 2 items have been thrown in. For this agent, there is no way of designing the Smokers without (directly or indirectly) testing the semaphore a. If the only operations we could perform on a are P_G and V, then irrespective of the other semaphores, variables and their operations, the controls of the Smokers cannot be designed.

The argument is very simple. Suppose that when Wrapper and Match are thrown in, if the Smoker with Tobacco can pick them up (correctly) by following a certain path; then even when Match and Tobacco are thrown in, by following exactly the earlier path, the Smoker with Tobacco can pick them up, leading to a contradiction. This is a trivial application of Lemma 2.

We agree with Parnas [6] that it is unrealistic to fix certain control sections. In the 2 Producer 2 Consumer problem, we let the designer have a free hand on every producer and consumer so that the system becomes autonomous.

Finally, from our limited experience, we could perceive that there are dangers involved in the usage of P's and V's, perhaps even more than those encountered when using GoTo's in sequential programs. Hansen's [4] structured approach seems to be a step in the right direction. We are presently developing structures for multi-programming from a different perspective.

ACKNOWLEDGMENTS

Thanks are due to Professor M.J. Flynn for his many helpful comments. The author is grateful to Mr. Lee Hoevel for a careful reading of the manuscript.

REFERENCES

1. P.J. Denning, Third Generation Computer Systems, Computing Surveys, Vol. 3, 1971, pp. 175-216.

2. E.W. Dijkstra, Cooperating Sequential Processes, in Programming Languages, ed. F. Genuys, Academic Press, 1968.

3. N. Habermann, On a solution and a generalization of the Cigarette Smokers' Problem, Tech. Report, Carnegie-Mellon University, August 1972.

4. P.B. Hansen, A Comparison of Two Synchronizing Concepts, Acta Informatica, 1972, pp. 190-199.

5. S.R. Kosaraju, Limitations of Dijkstra's semaphore primitives and Petri nets, Tech. Report #25, The Johns Hopkins University, Maryland, May 1973.

6. D.L. Parnas, On a solution to the Cigarette Smokers' Problem (without conditional statements), Tech. Report, Carnegie-Mellon University, July 1972.

7. S.S. Patil, Coordination of Asynchronous Events, Project MAC TR-72, June 1970.

8. S.S. Patil, Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes, Project MAC Memo 57, February 1971.