A Protocol of Mutual Exclusion for DSM Based on Vectors of Global Timestamps

Ludwik Czaja
[1]Vistula University, Warsaw
[2] Institute of Informatics, The University of Warsaw
lczaja@mimuw.edu.pl

Abstract

A new protocol using vectors of global timestamps for mutual exclusion in systems with Distributed Shared Memory (DSM) is described and some of its properties proved.

1. Introduction

Exclusive access to resources in concurrent programming has found various solutions originating in aforetime works by Dekker (unpublished but presented in [Dij 1968]), Dijkstra [Dij 1968], [Dij 2002], Lamport [La 1978], [La 1979], Ricart &Agrawala [R-A 1981], Saxena&Rai [S-R 2003] and a number of others. With coming of real multicomputer distributed systems without central memory and clock, where cooperation or competition of computers takes place only by message passing, the problem became essentially more complicated than in case of time-sharing systems or multiprocessors with shared physical memory. This concerns especially systems with no aid of central server: the computers „negotiate" by exchanging messages through the network and only one at a time is entitled to access a resource. A protocol based on vectors of global timestamps is presented in this paper and some of its properties are proved. The protocol is intended for systems with distributed shared memory (DSM), where the local memory in every computer is uniformly accessible for all computers: DSM is treated as a union of local memories. It is known that applications using DSM with some models of memory consistency [Cz 2016], require mutual exclusion. In the described solution, every computer keeps a vector of global timestamps of current requests for critical section, which are being issued by the connected computers. We do not discuss in details problems of timestamp advancement and mechanism of vector clocks (cf, for instance [S-R 2003], [K-M-C-H 2016]). It is assumed that such mechanisms provide current values of timestamps for the protocol described here. Likewise an issue of deadlock and fairness has been omitted, because of permitted space limitation of this paper.

A schematic structure of multicomputer system with Distributed Shared Memory is in Fig.1.1.

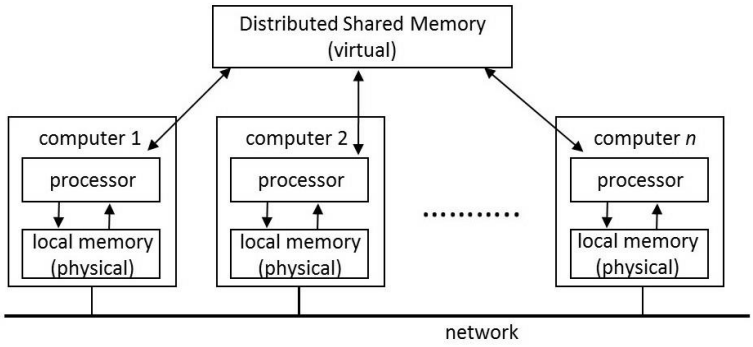


Fig.1.1.

2. Global timestamps revisited

A set Z is partially ordered iff its elements are related by relation $\sqsubseteq \subseteq Z \times Z$ satisfying: for every $x \in Z,\ y \in Z,\ z \in Z$:

1. $x \sqsubseteq x$ (reflexivity)
2. if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$ (antisymmetry)
3. if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$ (transitivity)

Moreover, if apart from (1), (2), (3):

4. $x \sqsubseteq y$ or $y \sqsubseteq x$ (connectivity)

then Z is linearly (totally) ordered. As usually, $x \sqsubset y$ iff $x \sqsubseteq y$ and $x \neq y$

Let $E(S)$ denote a set of events that may occur during activity of a distributed system S and let us define a partial order relation combining two kinds of precedence of events: occurring in the same process and of message sending and reception. For $x, y \in E(S)$ two auxiliary binary relations $\xrightarrow{process}$ and $\xrightarrow{message}$ are admitted as primary notions with the meaning:

- if x precedes y in the same process or if $x = y$ then $x \xrightarrow{process} y$
- if x is a sending message by a certain process and y is a reception of this message by another process then $x \xrightarrow{message} y$

A (weak) precedence $\xrightarrow{precedes} \subseteq E(S) \times E(S)$ is the least relation satisfying:

- if $x \xrightarrow{process} y$ or $x \xrightarrow{message} y$ then $x \xrightarrow{precedes} y$
- if $x \xrightarrow{precedes} y$ and $y \xrightarrow{precedes} z$ then $x \xrightarrow{precedes} z$

Events x, y are independent (concurrent) iff neither $x \xrightarrow{precedes} y$ nor $y \xrightarrow{precedes} x$, written $x||y$

Relation $\xrightarrow{precedes}$ is a modified precedence introduced by Lamport [La 1978] but due to the reflexivity of $\xrightarrow{process}$, the relation $\xrightarrow{precedes}$ is a partial order – contrarily to the Lamport's version. A common (global) clock and common memory are absent in asynchronous distributed systems, and partially ordered events are watched from outside of the system as occuring in real (external) time, thus, their precedence relation should imply similar order between real time instants of their occurrences. So, an injection mapping $C : E(S) \rightarrow R$ (R - set of real numbers), called a *logical clock* should be defined, satisfying implication $x \xrightarrow{precedes} y \Rightarrow C(x) \leq C(y)$, where values $C(x), C(y)$ are logical (not real) time instants of events x, y. To avoid absurd relationship of events to their time instants visible from outside of the system (when message reception precedes its dispatch), a compensation of processors' local clocks is necessary. So, if a sender sends a message together with its local time of dispatch and a receiver gets it earlier with respect to its local time, then the receiver must put forward its clock (a time register) to a time a little later than the time received from the sender. This procedure ensures the implication $x \xrightarrow{precedes} y \Rightarrow C(x) \leq C(y)$ if the values $C(x), C(y)$ are assumed to be compensated time instants of events x, y, called their *timestamps*. Obviously the reverse implication does not hold for some x, y if $x||y$ (see Fig.2.1). The logical clock C measures the compensated time. But it may happen that $C(x) = C(y)$ and $x \neq y$ for some concurrent x, y, so the mapping C is not one-to-one function, thus C does not establish unique representation of events by their timestamps. However if the processes are linearly ordered, e.g. numbered and the notion of event's timestamp is supplemented with a number of the process in which the event appears, then events can be uniquely represented by the richer timestamps, called *global*. So, let $nr(p_x)$ be a unique number of process p_x in which the event x occurs (a given event may occur in exactly one process, thus it identifies this process). A pair $\langle C(x), nr(p_x) \rangle$ is called a *global timestamp* of event x and let $\preccurlyeq$ denote a relation between global timestamps defined as $\langle C(x), nr(p_x) \rangle \preccurlyeq \langle C(y), nr(p_y) \rangle$ iff $C(x) < C(y)$ or if $C(x) = C(y)$ then $nr(p_x) \leq nr(p_y)$. Obviously $\preccurlyeq$ is linear, the so-called lexicographic order and the one-to-one injection mapping $\Gamma : E(S) \rightarrow R \times N$ (N - set of natural numbers) has been established by $\Gamma(x) = \langle C(x), nr(p_x) \rangle$, because $\Gamma(x) = \Gamma(y) \Rightarrow x = y$ for all x, y. Therefore, Γ establishes a unique representation of events by their global timestamps. Again, the implication $x \xrightarrow{precedes} y \Rightarrow \Gamma(x) \preccurlyeq \Gamma(y)$ holds but not the reverse one (see Fig.2.1).

Fig.2.1 exemplifies some relationships between events and their global timestamps during a system activity fragment. Black and grey circles are events of send and receive message respectively. Processes are numbered as follows: $nr(p1) < nr(p2) < nr(p3)$.

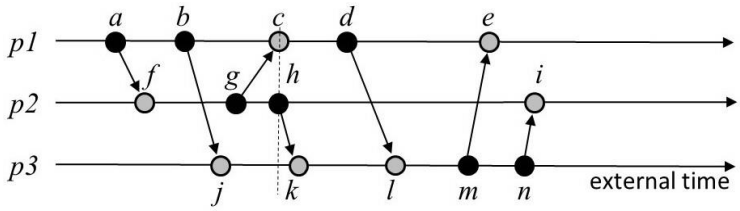


Fig.2.1. $a \xrightarrow{precedes} k \Rightarrow \langle C(a), nr(p_a)\rangle \prec \langle C(k), nr(p_k)\rangle$, $\langle C(c), nr(p_c)\rangle \prec C(h), nr(ph)\rangle$ but $c||h$ where $p_a = p_b = p_c = p_d = p_e = p1$. $p_f = p_g = p_h = p_i = p2$. $p_j = p_k = p_l = p_m = p_n = p3$.

3. Distributed mutual exclusion, a protocol and its properties

The global timestamps are used in a number of implementations of mechanisms in distributed systems. Consider a new protocol implementing distributed mutual exclusion with the following assumptions:

1. computers work in paralel asynchronously and are numbered $1, 2, ..., n$;
2. writing and reading to/from DSM memory is governed by the memory manager of each computer; computers communicate by message passing only and message propagation delay is finite but unpredictable.
3. there is one critical section (if there were more of them, the main concept of the protocol would be retained);
4. each request to the protocol for the critical section makes deliver of a current global timestamp of this event to the requesting computer;
5. computer of number i keeps vector $\overrightarrow{\mathbf{r}_i} = [r_{i1}, r_{i2}, ..., r_{in}]$ of variables r_{ik} $i, k = 1, 2, ..., n$ allocated in its physical memory; it stores the current timestamp of request in the component r_{ii} , then fetches values of components r_{kk} $(k \neq i)$ from remaining computers and stores them in variables r_{ik} of its vector $\overrightarrow{\mathbf{r}_i}$. Fig.3.1 depicts location of vectors of timestamps in the local memories;
6. initially all variables r_{ij} contain ∞ with $\infty > x$ for any number x;
7. by $min(\overrightarrow{\mathbf{r}_i})$ is denoted the least value of the components in $\overrightarrow{\mathbf{r}_i}$;

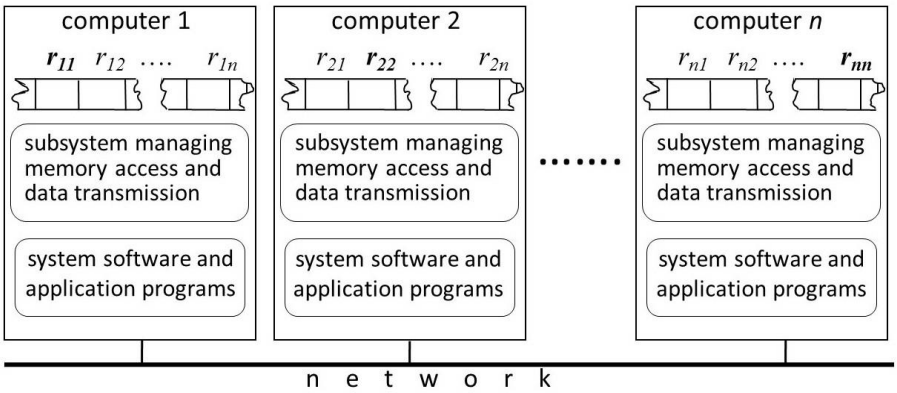


Fig.3.1. Structure of distributed system of n computers with vectors $\overrightarrow{\mathbf{r}_i} = [r_{i1}, r_{i2}, ..., r_{in}]$ $(i = 1, 2, ..., n)$ of timestamps allocated in local memories

A computer of number i when using the protocol depicted as a transition graph in Fig.3.2 for exclusive access to a protected resource, passes throughout the following states (subscript i in the names of states is omitted in order not to overload notation):

- W – execution of local (not critical) section
- B – import of current timestamps stored in variables r_{kk} of remaining computers; execution of $n-1$ assignments $r_{ik} := r_{kk}$ $(k \neq i)$; test of condition $r_{ii} > min(\overrightarrow{\mathbf{r}_i})$. State B is *stable* when the computer has completed fetch of all values of r_{kk} (note the various transmission duration of these values - see Theorem 3.2). In what follows, the adjective "stable" will be ommited when the noun "state" is used.
- Y – refusal to perform critical section (waiting state)
- R – execution of critical section
- G – release of critical section. This state is stable when the computer has completed broadcast of ∞ to all remaining computers.

Their set: $\mathbf{\Omega} = \{W, B, Y, R, G\}$

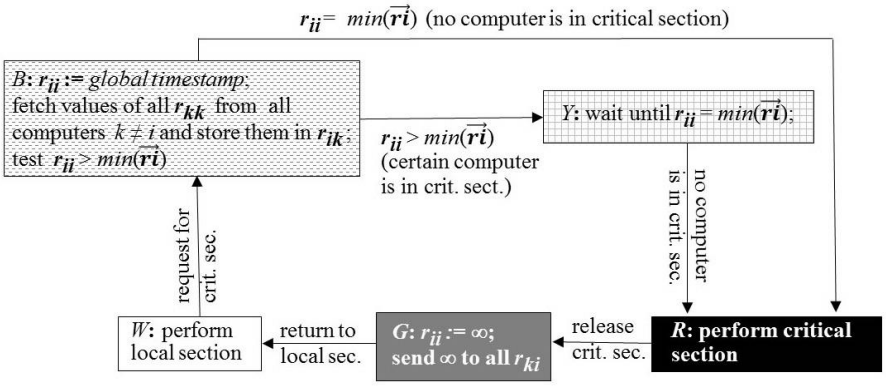


Fig.3.2. The distributed mutual exclusion protocol performed by computer of number i in the cycle from request for critical section till release.

Let us admit the following denotations:

- $Q_i \to Q_i'$ - computer of number i passes from a state $Q_i \in \mathbf{\Omega}$ to the next state $Q_i' \in \mathbf{\Omega}$ in the transition graph depicted in Fig.3.2. Note that transitions $B \to R$ and $Y \to R$ are possible when $r_{ii} = min(\overrightarrow{r_i})$, thus due to steady growth of global timestamp as a strictly increasing function of time, at most one computer may perform critical section at a time. A formal proof is given further.

- Set of global states $\mathbf{\Omega}^n = \underbrace{\mathbf{\Omega} \times \mathbf{\Omega} \times ... \times \mathbf{\Omega}}_{n \text{ times}}$ (the ith component correspods to computer number i) satisfying: if $\overrightarrow{Q} = [Q_1, Q_2, \ldots, Q_n] \in \mathbf{\Omega}^n$ then $\neg\exists i, j : (i \neq j \wedge Q_i = R \wedge Q_j = R)$.

- Initial state: $\overrightarrow{Q_{init}} = [W, W, \ldots, W] \in \mathbf{\Omega}^n$ with $r_{ij} = \infty$ for every computer $i = 1, 2, \ldots, n$.

- For $\overrightarrow{Q} = [Q_1, Q_2, \ldots, Q_n] \in \mathbf{\Omega}^n$ and $\overrightarrow{Q'} = [Q_1', Q_2', \ldots, Q_n'] \in \mathbf{\Omega}^n$ let $\overrightarrow{Q} \rightrightarrows \overrightarrow{Q'}$ mean: there exists a computer of number i such that $Q_i \to Q_i'$ and if $\neg Q_j \to Q_j'$ then $Q_j = Q_j'$. $\overrightarrow{Q'}$ is the next global state following $\overrightarrow{Q}$. As usually, by $\overrightarrow{Q} \overset{*}{\rightrightarrows} \overrightarrow{Q'}$ is denoted reachability of $\overrightarrow{Q'}$ from $\overrightarrow{Q}$ i.e existence of global states $\overrightarrow{Q^0}, \overrightarrow{Q^1}, ..., \overrightarrow{Q^{m-1}}, \overrightarrow{Q^m}$ with $\overrightarrow{Q^0} = \overrightarrow{Q}$, $\overrightarrow{Q^m} = \overrightarrow{Q'}$, $\overrightarrow{Q^j} \rightrightarrows \overrightarrow{Q^{j+1}}$, $j = 0, 1, ..., m-1$.

It follows from the transition graph in Fig.3.2 that for any computer of number $i = 1, 2, ...n$:

1. Storing a timestamp in register r_{ii} proceeds only in the state B of computer of number i; r_{ii} retains this value until the transition $R \to G$ takes place.

2. Storing ∞ in register r_{ii} and sending to r_{ki} of remaining computers proceeds only in the state G. Thus, from point 1 follows that r_{ii} decreases its value only in the state B.

3. Global states are exactly those reachable from the initial state $\overrightarrow{Q_{init}} = [W, W, \dots, W]$.

4. Because computation of $min(\overrightarrow{\mathbf{r}_i})$ in the state B of computer of number i takes place on completion of fetching values of r_{kk} from remaining computers, the order of entering computers into the critical section does not depend of the transmission latency. This is the FCFS order (First Come First Served) due to the steady growth of the global timestamps. Formal proofs of mutual exclusion realized by the protocol in Fig.3.2 as well as independence of the FCFS strategy of the order of message transmissions and their latency when executing actions in the state B are given in Theorems 3.1 and 3.2.

Table 3.1 presents an exemplary piece of run of a four computers system with Distributed Shared Memory. This is the following succession of global states:
$[B, W, B, W] \rightrightarrows [Y, W, R, W] \rightrightarrows [Y, W, R, B] \rightrightarrows [Y, B, R, Y] \rightrightarrows [Y, Y, G, Y] \rightrightarrows [R, Y, W, Y] \rightrightarrows [G, Y, W, Y] \rightrightarrows [W, Y, W, R]$
Global timestamps, i.e. pairs of numbers, are coded by single numbers – for simplicity of notation.

Before demonstration of correctness of the protocol and its FCFS strategy of giving entrance to critical section for computers, let us make some remarks.

- **Consumption of time**. In the state B, computer i, on request for critical section, broadcasts message „send me value of your r_{kk}" to all $n-1$ remaining computers, and waits for delivery. In the worst case the message reaches all destinations one after one and responses arrive one after one. This takes $2(n-1)$ transmissions. In the state G, on release of critical section, the computer i broadcasts ∞ to all r_{ki} of all $n-1$ remaining computers. This takes $n-1$ transmissions in the worst case.

- **Failure**. If a computer k sends incorrect timestamp stored in r_{kk} to requesting computers in the state B, then their behaviour depends on this value. This may cause indefinite wait of requesting computer i in the state Y (if r_{kk} is small enough to make $min(\overrightarrow{\mathbf{r}_i})$ permanently less than r_{ii}) or violation of mutual exclusion (if computer k delivers to computer i value of r_{kk} such that $r_{ii} = min(\overrightarrow{\mathbf{r}_i})$, and after a while it delivers to computer j value of r_{kk} such that $r_{jj} = min(\overrightarrow{\mathbf{r}_j})$; computer j may then enter into the critical section before computer i leaves it). Problems of failure as well as decidability of deadlock and fairness (cf. [Cz 1980]) are left to a separate paper.

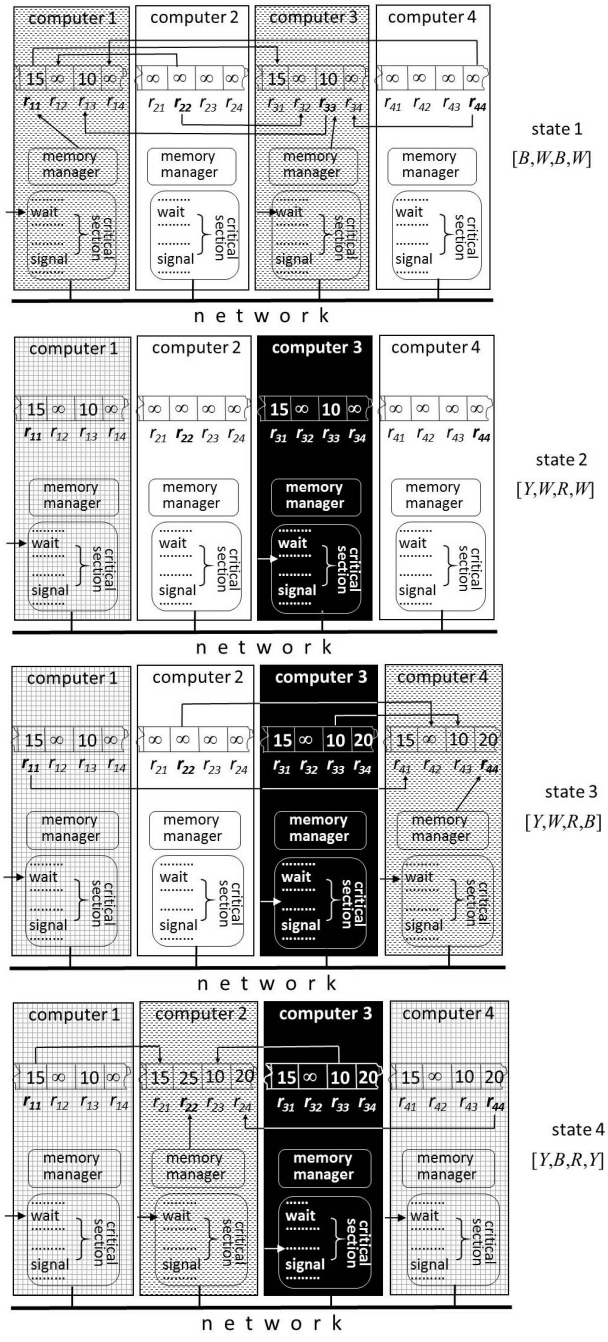
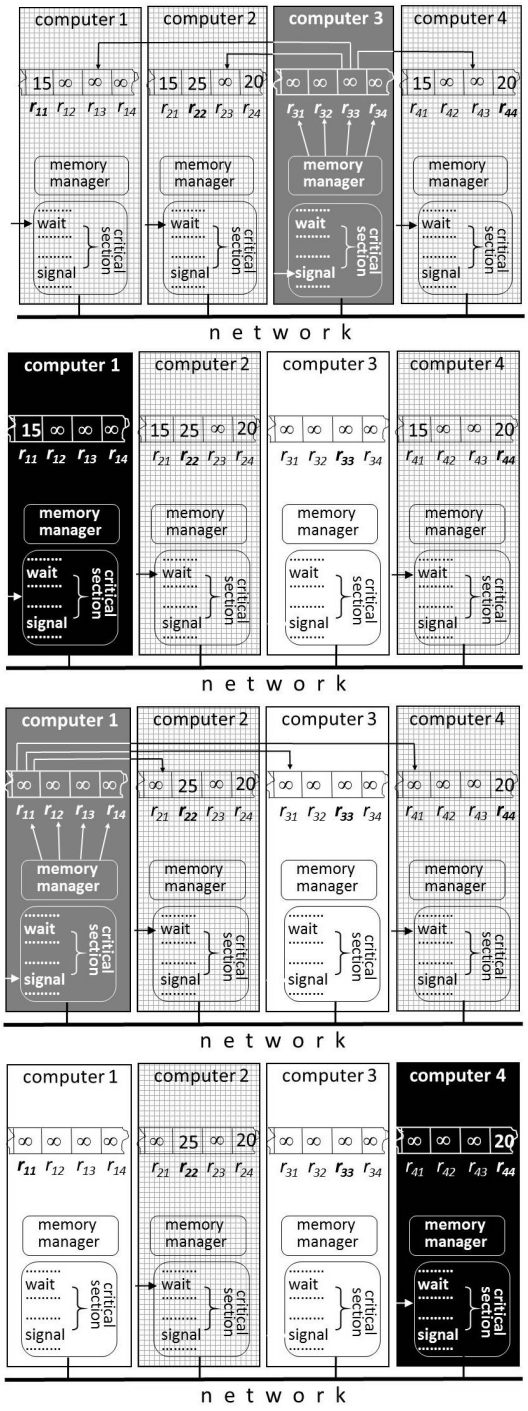


Table 3.1 Exemplary run of a system with four computers using protocol depicted in Fig.3.2. Pattern of the computers' background corresponds to their local states as pictured in the protocol in Fig.3.2



state 5
$[Y,Y,G,Y]$

state 6
$[R,Y,W,Y]$

state 7
$[G,Y,W,Y]$

state 8
$[W,Y,W,R]$

Table 3.1 cont.

Theorem 3.1

In no global state two distinct computers can perform critical section.

Proof. Let on the contrary, in a global state $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n] \in \mathbf{\Omega}^n$ computers of number i and j perform critical section. Then $r_{ii} = min(\overrightarrow{\mathbf{r}_i})$ and $r_{jj} = min(\overrightarrow{\mathbf{r}_j})$ in the local states Q_i and Q_j of these computers. By definition of the global timestamps $r_{ii} \neq r_{jj}$ because events of request for critical section are distinct, so, their global timestamps (i.e. values of r_{ii} and r_{jj}) are also distinct – due to the one-to-one function Γ (Section 3). But because of actions in the stable state B of the protocol in Fig.3.2, $r_{ij} = r_{jj}$ and $r_{ji} = r_{ii}$ hold. Since r_{ii} and r_{jj} are minimal in vectors $\overrightarrow{\mathbf{r}_i}$ and $\overrightarrow{\mathbf{r}_j}$ respectively, so, $r_{ii} \preceq r_{ij}$ and $r_{jj} \preceq r_{ji}$, therefore $r_{ii} \preceq r_{jj}$ and $r_{jj} \preceq r_{ii}$ which implies $r_{ii} = r_{jj}$ (by antisymmetry of $\preceq$ - see Section 2 for definition of the order $\preceq$ between global timestamps) – a contradiction! $\square$

Lemma 3.1

Suppose that in a global state $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n]$, computers of number i and j both are not in the local state W (i.e. $min(\overrightarrow{\mathbf{r}_i}) < \infty$, $min(\overrightarrow{\mathbf{r}_j}) < \infty$) or both are in W (i.e. $min(\overrightarrow{\mathbf{r}_i}) = \infty$, $min(\overrightarrow{\mathbf{r}_j}) = \infty$). Then $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$.

Proof. Let $min(\overrightarrow{\mathbf{r}_i}) < \infty$, $min(\overrightarrow{\mathbf{r}_j}) < \infty$. Then in the global state $\overrightarrow{Q}$, exactly one computer, say of number k, either is performing critical section or is ready to do this, therefore $r_{kk} = min(\overrightarrow{\mathbf{r}_k})$. According to the protocol in Fig.3.2 $r_{ik} = r_{jk} = r_{kk} = min(\overrightarrow{\mathbf{r}_k})$ and r_{ik}, r_{jk} are the least components of vectors $\overrightarrow{\mathbf{r}_i}$, $\overrightarrow{\mathbf{r}_j}$ in the state $\overrightarrow{Q}$. Thus $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_k}) = min(\overrightarrow{\mathbf{r}_j})$. $\square$

Theorem 3.2

Computers enter into the critical section in the order of their requesting for it. This order is independent of the data transmission latency.

Proof. Let $\overrightarrow{Q} = [Q_1, ...Q_i, ...Q_j, ...Q_n]$ be a global state with local states Q_i, Q_j each of them either B or Y, thus $r_{ii} < \infty$, $r_{jj} < \infty$. It is to be proved that if computer i enters into state Q_i before entering of j into state Q_j, i.e. if $r_{ii} < r_{jj}$, then state R would be reached by computer i before computer j. By Lemma 3.1, $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$ in $\overrightarrow{Q}$ and according to the protocol in Fig.3.2, variables r_{ii}, r_{jj} are not changing their values until computers i, j reach state G. Thus, if eventually a global state $\overrightarrow{Q'} = [Q'_1, ..., Q'_i = R, ...Q'_j, ...Q'_n]$ nearest to $\overrightarrow{Q}$ is reached from $\overrightarrow{Q}$ then $min(\overrightarrow{\mathbf{r}_i}) = r_{ii}$ in $\overrightarrow{Q'}$. But if a global state $\overrightarrow{Q"} = [Q"_1, ..., Q"_i, ...Q"_j = R, ...Q"_n]$ had been reached from $\overrightarrow{Q}$ before $\overrightarrow{Q'}$ then $min(\overrightarrow{\mathbf{r}_j}) = r_{jj}$ in $\overrightarrow{Q"}$ would hold. Thus, $min(\overrightarrow{\mathbf{r}_i}) \leq r_{ii} < r_{jj} = min(\overrightarrow{\mathbf{r}_j})$ because values of r_{ii} and of r_{jj} in the state $\overrightarrow{Q"}$ are the same as in $\overrightarrow{Q}$ and $min(\overrightarrow{\mathbf{r}_i}) = min(\overrightarrow{\mathbf{r}_j})$ - a contradiction!

Now, note that the value of $min(\overrightarrow{\mathbf{r}_i})$ is established only when values of r_{kk} are completely transmitted from all computers $k \neq i$ to computer i and stored in r_{ik} (see the protocol in Fig.3.2). This value does not depend on duration of these

transmissions nor on their order. Therefore the order of entering computers into critical section is independent of transmission latency but only on the order of their requesting for it. □

Independence of $min(\overrightarrow{\mathbf{r}_i})$ of order as well as latency of transmissions when the run of four computers system shown in Table 3.1, has reached state 1: $[B, W, B, W]$, is illustrated in Fig.3.3(a) and (b). $i\uparrow(r_{kk})_k$ means: "computer k sends value of r_{kk} up to computer i" ; $k\downarrow(r_{ik})_i$ means: "computer i receives a value sent by computer k and stores it in r_{ik}".

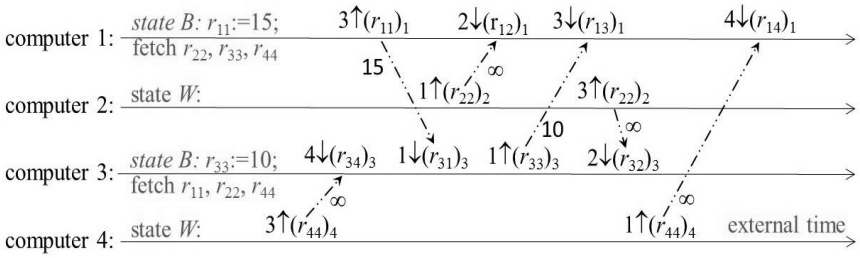


Fig.3.3(a). Diagram of global state $[B, W, B, W]$

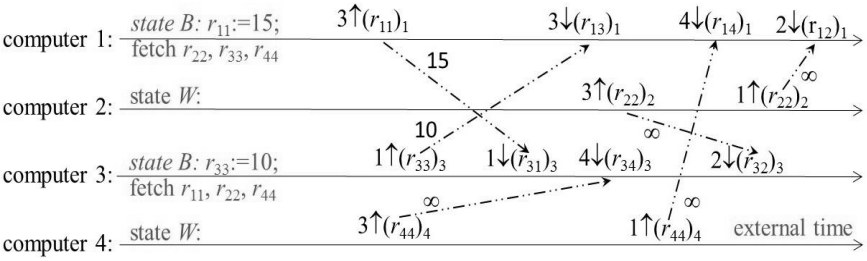


Fig.3.3(b). The same global state and $min(\overrightarrow{\mathbf{r}_i})$ as in (a) but different order and latency of transmissions

Summary

The protocol presented in Fig.3.2 may seem similar to that in [R-A 1981] in that it is fully distributed (without a coordinating server) and because of usage of global timestamps and two-way communication between computer requesting for critical section and remaining computers. However the algorithm proposed here is differently organized: a requesting computer, updates its own vector of timestamps and makes a decision on the basis of its content whether to enter into critical section or wait. Using ∞ as a largest number, not assumed by any timestamp unifies activities when making the decision. Most important properties of the algorithm are formally proved. The algorithm seems suitable for system with Distributed Shared Memory, since problems with data inconsistency do not arise (Theorem 3.2).

References

[Cz 1980] Czaja L., *Deadlock and Fairness in Parallel Schemas: a Set-Theoretic Characterization and Decision Problems*, Information Processing Letters, vol. 10 number 4,5 (1980)

[Cz 2016] Czaja L., *Remarks on Memory Consistency Description*, to appear in Fundamenta Informaticae 2016

[Dij 1968] Dijkstra E.W., *Cooperating sequential processes*, in F. Genuys, ed., Programming Languages: NATO Advanced Study Institute, pp. 43-112, Academic Press, 1968

[Dij 2002] Dijkstra E.W., *The Origin of Concurrent Programming,* (book) Springer Verlag New York, 2002 pp. 65-138

[K-M-C-H 2016] Ihor Kuz, Manuel M. T. Chakravarty & Gernot Heiser, *A course on distributed systems* COMP9243, 2016, 2016

[La 1978] Lamport L., *Time, Clocks and the Ordering of Events in Distributed Systems*, Comm. of the ACM, 1978, 21, pp. 558-565

[La 1979] Lamport L., *How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs*, IEEE Trans. On Computers, 1979 C-28, s. 690-691

[R-A 1981] Ricart G., Agrawala A.K., *An Optimal Algorithm For Mutual Exclusion in Computer Networks*, Comm. of the ACM, 1981, 24, 1, pp. 9-17

[S-R 2003] Saxena P.C., Rai J., A survey of permission-based distributed mutual exclusion algorithms, in Computer Standards & Interfaces, Volume 25, Issue 2, May 2003, Pages 159–181