数理解析研究所講究録 1205 巻 2001 年 59-64

Lockout Avoidance Algorithms without Using Time-Stamps for the k-Exclusion Problem

Michiko Omori, Kumiko Obokata, and Yoshihide Igarashi
(大森 透子) (小保方 久美子) (五十嵐 善英)

Department of Computer Science, Gunma University, Kiryu, Japan 376-8515
(群馬大学工学部)

E-mail:igarashi@comp.cs.gunma-u.ac.jp

Abstract

We propose two lockout avoidance algorithms for the k-exclusion problem without using time-stamps on asynchronous multi-writer/reader shared memory model. The first algorithm is a modification of the n-process algorithm for the mutual exclusion problem by Peterson or its variation. The second algorithms is a combination of the first algorithm and the tournament algorithm for the mutual exclusion problem by Peterson and Fisher. The correctness and efficiencies of these algorithms are shown.

key words: asynchronous processes, concurrent computation, k-exclusion, lockout avoidance, shared memory

1 Introduction

The k-exclusion problem is a natural extension of the mutual exclusion problem. In k-exclusion, some number of processes, specified by the parameter k, are allowed to be concurrently inside the critical region, where corresponding users can use the resource. The solution by Fischer et al.[5] can tolerate the slowdown or even the crash (i.e., stopping failure) of up to $k-1$ processes, but it was based on the use of a powerful primitive, *read-modify-write*. It is a *first-in, first-enable* solution to the k-exclusion problem. Afek et al.[1] gave another solution to the *first-in, first-enable* k-exclusion problem. Unlike the solution by Fischer et al. this solution does not use a powerful *read-modified-write* primitive. By use of a *concurrent time-stamp system* [2, 3, 4], it requires only bounded read/write shared memory [1].

In this paper we propose two algorithms without using concurrent time-stamps for the k-exclusion problem on the asynchronous multi-writer/reader shared memory model [8]. The first algorithm is a modification of the n-process algorithm for the mutual exclusion problem by Peterson [11] or a variation of the algorithm accelerated by Igarashi and Nishitani [7]. The second algorithm is a combination of the first algorithm and the tournament algorithm for the mutual exclusion problem by Peterson and Fisher [12]. We only consider the simplest type of process failures: stopping failure, whereby a process just stops without warning. Both the algorithms are immune to stopping failures of fewer than k processes. These algorithms are simple and lucid, but not *first-in, first-enable*. All logarithms in this paper are to the base 2. The shared memory size for the first algorithm is $(n-k)\lceil \log n \rceil + n\lceil \log(n-k+1) \rceil$ bits. We describe our algorithms in terms of I/O automata [8, 10]. In order to estimate upper bounds on the running times of the algorithms, we impose an upper bound of l on the time between

successive atomic steps of each faultless process in the trying region and the exit region, and an upper bound of c on the time that any user spends in its critical region. The running time by the first algorithm and the running time by the second algorithm for the trying regions are bounded by $(n-k)c + O(n(n-k)^2)l$ and $(\frac{n}{k})^k c + O((\frac{n}{k})^{k+1}k)l$, respectively.

2 Preliminary

The computation model used in this paper is the asynchronous multi-writer/reader shared memory model. It is a collection of processes and shared variables. Interactions between a process and its corresponding user are by input actions from the user to the process and by output actions from the process to the user. The model is called an I/O automaton [9].

A user with access to the resource is modeled as being in a critical region. When a user is not involved in the resource, it is said to be in remainder region. In order to gain admittance to its critical region, a process executes a trying protocol. The duration from the start of executing the trying protocol to the entrance of the critical region is called the trying region. After the end of the use of the resource by a user, the corresponding process executes an exit protocol. The duration of executing the exit protocol is called the exit region. These procedures can be repeated in cyclic order, from its remainder region to its trying region, then to its critical region, then to its exit region, and then back again to its remainder region. The k-exclusion problem is to devise protocols for at most k processes to be concurrently in the critical region.

We assume that the n processes are numbered $1, \cdots, n$. Each process i corresponds to user U_i $(1 \leq i \leq n)$. The inputs to process i from user U_i are try_i which means a request by user U_i for access to the resource, and $exits_i$ which means an announcement of the end of the use of the resource by U_i. The output from process i to user U_i are $crit_i$ which means the grant of the resource to U_i, and rem_i which tells U_i that it can continue with the rest of its work. These are external actions of the shared memory system.

The system to solve the k-exclusion problem should satisfy the following conditions.

(1) There is no reachable system state in which more than k processes are in the critical region.

(2) If at least one faultless process is in the trying region and less than k processes are in the critical region, then at some later point some process enters the critical region.

(3) If a faultless process is in the exit region, then at some later point the process enters the remainder region.

(4) If all users always return the resource and if the number of faulty processes of the stopping type is at most $f < k$, then any faultless process wishing to enter the critical region eventually does so.

Conditions (1), (2), (3) and (4) above are called k-exclusion, progress for the trying region, progress for the exit region, and k-lockout avoidance, respectively.

3 A lockout avoidance algorithm with $n - k$ levels

Previous algorithms on the shared memory model for the k-exclusion problem use concurrent time-stamp systems [1, 5, 6]. We propose a lockout avoidance algorithm without using time-stamps on the multi-writer/reader share memory model. In the n-process algorithm by Peterson [7, 11] for the mutual exclusion problem, to move process i from level s to level $s+1$ the condition $[\forall j \neq i : flag(j) < s]$ or $[turn(s) \neq i]$ should be satisfied. We modify the first part of the condition in the waitfor statement.

```
procedure (n,k)-EXCL
shared variables
  for every s ∈ {1,...,n − k}:
    turn(s) ∈ {1,...,n}, initially arbitrary, writable and readable by all processes;
  for every i ∈ {1,...,n}:
    flag(i) ∈ {0,...,n − k}, initially 0, writable by i and readable by all j ≠ i;

process i
  input actions {inputs to process i from user U_i}:
    try_i, exit_i;
  output actions {outputs from process i to user U_i}:
    crit_i, rem_i;

  ** Remainder region **
  try_i:
    for s = 1 to n − k do
      begin
        flag(i) := s;
        turn(s) := i;
        waitfor [|{j|j ≠ i : flag(j) ≥ s}| ≤ n − s − 1] or [turn(s) ≠ i]
      end;
  crit_i;
  ** Critical region **
  exit_i:
    flag(i) := 0;
  rem_i;
```

For each level s ($1 \leq s \leq n-k$), statement waitfor$[|\{j|j \neq i : flag(j) \geq s\}| \leq n-s-1]$ or $[turn(s) \neq i]$ in (n,k)-*EXCL* is not atomic step. It consists of a number of atomic steps for checking the condition by process i at level s. This means that for each s, the contents of the local variable *count* does not necessarily represent the number of other processes that locate at level s or above. Since the shared memory used in this paper is not the *read-modified-write* model, this uncertainty cannot be avoided. Nevertheless the uncertain information about the number of processes at level s or above in the local variable *count* together with the contents of shared memory $turn(s)$ is good enough to guarantee k-exclusion as shown in Section 4.

4 Correctness for (n,k)-*EXCL*

For a faultless process, either it takes an infinite number of steps or it ends in the remainder region. We assume that the number of process failures is always at most $k-1$. In an execution by (n,k)-*EXCL*, process i is said to be a winner at level s if it has left the waitfor statement in the sth loop of the **for** statement. Note that if a process is a winner at level s then for any $1 \leq t \leq s$, the process is also a winner at level t. For each i ($1 \leq i \leq n$), when process i has entered its exit region, the qualification for the winner of process i is cancelled. The proofs of the lemmas and the theorems in this section are omitted due to the page limit.

Lemma 1 *In any reachable system state of* (n,k)-*EXCL, the number of winners at level 1 is less than* n.

Lemma 2 *In any reachable system state of* (n,k)-*EXCL, for any* $1 \leq s \leq n-k$ *there are at most* $n-s$ *winners at level* s.

Theorem 3 *(n,k)-EXCL guarantees k-exclusion even if any number of process failures of the stopping type exist.*

In order to prove progress for the trying region and k-lockout avoidance, it is enough to give a time bound for the trying region in the case where at most $k-1$ stopping failures of processes exist.

Theorem 4 *Suppose that the number of stopping failures of processes is always at most $k-1$. In (n,k)-EXCL, the time from when a faultless process enters its trying region until the process enters its critical region is at most $(n-k)c + O(n(n-k)^2)l$.*

The following theorem is from Theorem 3 and Theorem 4.

Theorem 5 *(n,k)-EXCL solves the k-exclusion problem, and it is k-lockout avoidance.*

The running time of (n,k)-$EXCL$ in the trying region for a process is $(n-k)c+O(n(n-k)^2)l$. It is possible to construct a scenario of process behavior that leads the running time of a process in the trying region as slow as the running time bound given in Theorem 4. The running time of (n,k)-$EXCL$ is not good compared with other methods. However, for a large k such as $n-k=O(1)$ or $n-k=o(n)$, the algorithm is fast.

5 A group tournament algorithm

In this section, for the technical reason we number the processes as $0,\cdots,n-1$ rather than $1,\cdots,n$. We denote the null sequence by λ. For a set of processes G, the lower half of G is the set of $\lceil |G|/2 \rceil$ lower numbered processes of G, and the upper half of G is the set of $\lfloor |G|/2 \rfloor$ higher numbered processes of G.

Let $T^{(n,k)}$ be the complete binary tree with depth $\lfloor \log(n/(k+1)) \rfloor$. Such a tree is called a tournament tree. Each node of $T^{(n,k)}$ is labeled by the following rule:

(1) The root of the tree is labeled by λ.
(2) The left son and the right son of a node with label x are labeled by $x0$ and $x1$, respectively.

Let $bl(n,k) = \lfloor \log(n/(k+1)) \rfloor$. For each binary sequence x with a length not longer than $bl(n,k)$, G_x is a set of processes defined as follows: $G_\lambda = \{0,\cdots,n-1\}$. If $x = x'0$ then G_x is the lower half of $G_{x'}$, and if $x = x'1$ then G_x is the upper half of $G_{x'}$. For each node x of $T^{(n,k)}$, the node x is associated with the set of processes G_x. For integer pairs, (i,j)'s, we use the lexicographic order. For each pair of $0 \le i \le n-1$ and $0 \le t \le bl(n,k)$, $g(i,t)$ is defined to be x if $|x| = t$ and i belongs to G_x. From the definitions of G_x and $bl(n,k)$, $g(i,t)$ is a well defined function on $\{(i,t) | 0 \le i \le n-1, 0 \le t \le bl(n,k)\}$.

For a leaf x of $T^{(n,k)}$, a process i in G_x joins the competition for G_x whenever i enters its trying region. We apply the technique used in procedure (n,k)-$EXCL$ to the competitions. Each process engages in a series of $k \cdot bl(n,k) + |G_{g(i,bl(n,k))}| - k$ competitions. Then the number of final winners of the competitions for G_x is at most k. For an inner node x of $T^{(n,k)}$, final winners from G_{x0} and G_{x1} can join the competitions for G_x. Then for these winners from the competitions for G_{x0} and G_{x1} we apply again the technique used in procedure (n,k)-$EXCL$ to the competitions for G_x. Repeating this way, we can choose at most k final winners of the competition for G_λ. These final winners are allowed to enter their critical regions.

Hereafter, for simplicity we assume that $n = 2^t(2k)$ for a nonnegative integer t. For a general value for n, we need only minor modifications. The following procedure (n,k)-$GTEX$ is the group tournament algorithm for the k-exclusion problem, where $n = 2^{bl(n,k)}(2k)$.

procedure (n,k)-$GTEX$
shared variables
for every (x, s) such that $0 \leq |x| \leq bl(n,k), 1 \leq s \leq k$:
$turn(x, s) \in G_x$, initially arbitrary, writable and readable by all processes in G_x;
for every $i \in G_\lambda$:
$flag(i) \in \{(a,b)|0 \leq a \leq bl(n,k), 1 \leq b \leq k\}$,
initially $(0,0)$, writable by i and readable by all $j \neq i$;

process i
input actions {inputs to process i from user U_i}:
try_i, $exit_i$;
output actions {outputs from process i to user U_i}:
$crit_i$, rem_i;

** Remainder region **
try_i:
for $p = 0$ **to** $bl(n,k)$ **do**
for $q = 1$ **to** k **do**
begin
$flag(i) := (p,q)$;
$turn(g(i, bl(n,k) - p), q) := i$;
waitfor $[|\{j|j \neq i, j \in G_{g(i,bl(n,k)-p)} : flag(j) \geq (p,q)\}| \leq 2k - q - 1$, or
$turn(g(i, bl(n,k) - p), q) \neq i]$
end;
$crit_i$;
** Critical region **
$exit_i$:
$flag(i) := (0,0)$;
rem_i;

6 Correctness for (n,k)-$GTEX$

In an execution of (n,k)-$GTEX$, process i is said to be a winner at (p,q) for $G_{g(i,bl(n,k)-p)}$ if the process has left the waitfor statement in the qth inner **for** loop of the pth outer **for** loop. If a process i is a winner at (p,q) for $G_{g(i,bl(n,k)-p)}$, then for any $(p',q') \leq (p,q)$ the process is also a winner at (p',q') for $G_{g(i,bl(n,k)-p')}$. For each i $(0 \leq i \leq n-1)$, when process i has entered its exit region, the qualification as a winner for process i is cancelled. In an execution by (n,k)-$GTEX$, if we say a competition at (p,q), it means a competition at level q of a group competition in G_x associated with a node in depth $bl(n,k) - p$ of $T^{(n,k)}$. The proofs of the lemmas and the theorems in this section are omitted due to the page limit.

Lemma 6 *In any reachable system state of* (n,k)-$GTEX$, *for any* (p,q) $(0 \leq p \leq bl(n,k))$ *and any binary sequence* x *of length* $bl(n,k) - p$, *the number of winners in* G_x *at* (p,q) *is at most* $2k - q$.

Theorem 7 (n,k)-$GTEX$ *guarantees* k-*exclusion even if any number of process failures of the stopping type exist.*

Theorem 8 *Suppose that the number of stopping failures of processes is always at most* $k-1$. *In* (n,k)-$GTEX$, *the time from when a faultless process enters its trying region until the process enters its critical region is at most* $(\frac{n}{k})^k c + O((\frac{n}{k})^{k+1} k)l$.

Theorem 9 (n,k)-$GTEX$ *solves the* k-*exclusion problem, and it is* k-*lockout avoidance.*

7 Concluding remarks

We have proposed a lockout avoidance algorithms for the k-exclusion problem without using concurrent time-stamps on asynchronous multi-writer/reader shared memory model. In a execution by (n,k)-$EXCL$ or (n,k)-$GTEX$, each process observes the number of winners at a level, but its observation is somewhat uncertain. We showed that such uncertain information is still useful to guarantee k-exclusion and k-lockout avoidance. Each of two algorithms given in this paper, (n,k)-$EXCL$ and (n,k)-$GTEX$ has a simple structure and can be easily implemented although its performance in time and space is not best. We are interested in a problem whether some modification of these algorithms can reduce the shared memory size. We are also interested in a problem whether we can speed up our algorithm by some modification. The k-assignment problem is closely related to the k-exclusion problem. These problems would be worthy of further investigation.

References

[1] Y.Afek, D.Dolev. E.Gafni, M.Merritt, and N.Shavit, "A bounded first-in, first-enable solution to the l-exclusion problem", ACM Transactions on Programming Languages and Systems, vol.16, pp.939–953, 1994.

[2] H.Attiya and J.Welch, "Distributed Computing: Fundamentals, Simulations and Advanced Topics", McGraw-Hill, New York, 1998.

[3] D.Dolev and N.Shavit, "Bounded concurrent time-stamp systems are constructible", *21st Annual ACM Symposium on the Theory of Computing*, New York, pp.454–465, 1989.

[4] C.Dwork and O.Waarts, "Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible", *24th Annual ACM Symposium on the Theory of Computing*, Victoria, British Columbia, pp.655–666, 1992.

[5] M.J.Fischer, N.A.Lynch, J.E.Burns, and A.Borodin, "Resource allocation with immunity to limited process failure", *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, pp.234–254, 1979.

[6] M.J.Fischer, N.A.Lynch, J.E.Burns, and A.Borodin, "Distributed FIFO allocation of identical resources using small shared space", *ACM Transactions on Programming Languages and Systems*, vol.11, pp.90–114, 1989.

[7] Y.Igarashi and Y.Nishitani, "Speedup of the n-process mutual exclusion algorithm", Parallel Processing Letters, vol.9, pp.475–485, 1999.

[8] N.A.Lynch, "Distributed Algorithms", *Morgan Kaufmann*, San Francisco, California, 1996.

[9] N.A.Lynch and M.J.Fischer, "On describing the behavior and implementation of distributed systems", *Theoretical Computer Science*, vol.13, pp.17–43, 1981.

[10] N.A.Lynch and M.R.Tuttle, "Hierarchical correctness proofs for distributed algorithms", *6th Annual ACM Symposium on Principle of Distributed Computing*, Vancouver, British Columbia, pp.137–151, 1987.

[11] G.L.Peterson, "Myths about the mutual exclusion problem", *Information Processing Letters*, vol.12, pp.115–116, 1981.

[12] G.L.Peterson and M.J.Fischer, "Economical solutions for the critical section problem in a distributed system", *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, Boulder, Colorado, pp.91–97, 1977.