Fundamentals of Distributed Systems

Mutual exclusion on "shared-memory" systems

Advanced techniques

Mitică Craus, Cristian Buțincu

"Gheorghe Asachi" Technical University of Iași

Summary

Introduction

- Fundamental problem: there is a frequent need to mediate the concurrent access to a resource on distributed systems.
- The mutual exclusion problem arises on groups of processing units that periodically access certain resources that cannot be simultaneously used by more than one processing unit (i.e. a printer).
- Each processing unit can execute a code segment called critical section in such a way that, at any given time, at most one processing unit executes the critical section (*mutual exclusion*).
- *If one or more processing units try to enter into critical section, one of them will eventually succeed, as long as no processing unit remains into critical section indefinitely.*
- The aforementioned property does not guarantee a success because even if a processing unit tries to enter into critical section, it can continuously be ignored by the other processing units.
- A stronger property, that eliminates *deadlock*, is *no lockout* or *no starvation*: *If a processing unit tries to enter into critical section, it will eventually succeed, as long as no processing unit remains into critical section indefinitely.*

Introduction - continuation

- The program of a processing unit is divided into the following sections:

Entry: the code that prepares the entry into the critical section;
Critical: the code that must pe protected against concurrent execution;
Exit: the code that executes after leaving critical section;
Remainder: the rest of the code.

Bakery algorithm - description

- Main idea: the processing units that wish to enter into critical section behave similar to the customers of a bakery.
- Each customer that arrives at the bakery gets a number = (1 + the biggest of all the other customer's numbers).
- The customer that has the smallest number will be served next.
- The number assigned to the customers that do not stand in the queue is 0 (0 is not considered as the smallest number).

Bakery algorithm - description (continuation)

- Each processing unit p_i that tries to enter into critical section will consider its number as being 1 + (the biggest of all the other processing units' numbers).
  - The computed number will be recorded on the location *Number*[*i*] of *Number* array.
- Because many processing units can concurrently read the *Number* array, it is possible for some of them to compute the same number.
  - In order to avoid this, p_i's ticket is defined as the pair $(Number[i], i)$.
  - Thus, the processing units' tickets that try to enter the critical section are unique.
  - The tickets are ordered lexicographically.
- After computing its number, p_i waits until his ticket has the smallest value. The waiting process is as follows:
  1. p_i chooses the processing unit $p_j, j \in \{0, 1, \ldots, n-1\}$, where j is minimum and $j \neq i$.
  2. If p_j is in the process of computing its number, p_i waits for it to finish the computation and then it compares its ticket with p_j's ticket.
  3. If p_j's ticket is smaller than p_i's ticket, then p_i waits until p_j executes and leaves critical section.
  4. After this, p_i repeats the steps 2 and 3 for $p_{j+1}, p_{j+2}, \ldots, p_{n-1}, i \neq j+1, j+2, \ldots, n-1$.
- Algorithm disadvantage: if the situation where all processing units are into *remainder* section does not arise, the computed numbers can grow without limit.

Bakery algorithm - pseudocode

Notations:

- $Number[0..n-1]$ an array of n integers, that holds on location i the number computed by p_i,
- $Computing[0..n-1]$ an array of boolean values so that $Computing[i]$ is true as long as p_i is in the process of computing its number.

Premise: Initially, $Number[i]=0$ and $Computing[i]=false,\ i=0,1,\ldots,n-1$

Pseudocode for processing unit $p_i,\ i\in\{0,1,\ldots,n-1\}$

$\langle Entry\rangle$ /*p_i needs critical resource*/

$\text{B}\small{\text{AKERY}}\text{\_E}\small{\text{NTRY}}\text{\_E}\small{\text{M}}(Number, Computing, n, p_i)$

$Computing[i] \leftarrow true$ $/*$ p_i is in the process of computing its number $*/$

$Number[i] \leftarrow \max\{Number[0], Number[1], \ldots, Number[n-1]\}+1$ $/*$ computes the number $*/$

$Computing[i] \leftarrow false$ $/*$ p_i finishes computing its number $*/$

for $j \leftarrow 0$ *to* $n-1,\ j \neq i$

do $/*$ p_i waits until p_j finishes computing its number $*/$

  wait until $Computing[j] = false$

  $/*$ p_i waits until p_j's number is 0 or p_j's ticket $>$ its ticket $*/$

  wait until $Number[j]=0$ *or* $(Number[j], j) > (Number[i], i)$

$\langle Critical\rangle$ /*p_i enters critical section*/

$\langle Exit\rangle$ /*p_i exits critical section*/

$\text{B}\small{\text{AKERY}}\text{\_E}\small{\text{XIT}}\text{\_E}\small{\text{M}}(Number, Computing, n, p_i)$

$Number[i] \leftarrow 0$ $/*$ p_i relinquishes its number $*/$

$\langle Remainder\rangle$ /*p_i does not need critical resource*/

Correctness

Lemma (1)

If p_i is in critical section and $Number[k] \neq 0$, for an index $k \neq i$, then $(Number[k], k) > (Number[i], i)$.

Proof.

The processing unit p_i is in critical section. This means that it has finished the second *wait* (line 8) $(\forall) j \in \{0, 1, \ldots, n-1\}$. There are two situations:

1. p_i passed by line 8, for $j = k$, and $Number[k] = 0$. This means that when p_i finished the second *wait* for $j = k$, p_k was either in *reminder* section or it didn't finish computing its number. As p_i had finished the wait on line 6 for $j = k$, p_k could not have not finished computing its number; the wait on line 6 for $j = k$ ends by $Computing[k] = false$. Thus, p_k was in *remainder* section or it started computing its number after p_i had finished the second *wait*, for $j = k$. In the first case, lemma does not apply. In the second case, the number computed by p_k is greater than the number recorded by p_i on *Number* array, when the instruction on line 2 was executed; $Number[k] > Number[j]$, $(\forall) j \in \{0, 1, \ldots, n-1\}$. Thus, in the second case $(Number[k], k) > (Number[i], i)$.
2. p_i passed by line 8, for $j = k$, and $(Number[k], k) > (Number[i], i)$. Obviously, the inequality remains in place until p_i leaves critical section or p_k does not compute another number. If p_k computes another number, the inequality remains in place, because the number computed by p_k will be greater than $Number[j], (\forall) j \in \{0, 1, \ldots, n-1\}$.

□

Correctness - continuation

Corollary

A processing unit that is in critical section holds the smallest ticket among all the other processing units that try to enter into critical section.

Lemma (2)

It the processing unit p_i is in critical section, then $Number[i] > 0$.

Proof.

Before entering critical section, p_i computes $Number[i]$. From the computing formula, it results that $Number[i] > 0$. □

Correctness - continuation

Theorem (1)

Bakery algorithm ensures mutual exclusion.

Proof.

Let's assume, that at a certain moment, two processing units, p_i and p_j, are in the critical section. From lemma 2, it results that $Number[i] \neq 0$ and $Number[j] \neq 0$. Thus, lemma 1 is applicable, which states that $(Number[j], j) > (Number[i], i)$ and $(Number[i], i) > (Number[j], j)$. Impossible. $\square$

Correctness - continuation

Theorem (2)

The mutual exclusion offered by Bakery algorithm is no starvation.

Proof.

Let's assume that there are processing units that are *starving*, that is, they try to enter into critical section, but they don't succeed. Let p_i be the processing unit that *starves* and all the other processing units with smaller numbers do not *starve*. Obviously, all processing units that try to enter into critical section compute a number, because there is no blocking mechanism for computing the number that will be part of the ticket. All processing units that enter into *entry* section after p_i compute larger numbers, thus they won't enter into critical section before p_i. All processing units with smaller numbers will enter into critical section; we assumed that these don't *starve*. These will exit after a while from the critical section; no processing unit stays indefinitely into critical section. After their exit from critical section, p_i passes all tests and enters the critical section. This contradicts our assumption. □

Tournament algorithm - description

- The processing units are grouped in pairs and compete in a *tree-tournament* arrangement.
- The pairs arranged in a complete binary tree.
- Let $m = \lceil \log n \rceil - 1$ and T a complete binary tree with 2^m leaves (and a total of $n = 2^{m+1} - 1$ nodes).
- The tree nodes are numbered as follows:
  - The root is numbered as 1;
  - The left child of a v node is numbered with $2v$, and the right child with $2v+1$.
- It follows that the tree's leaves are numbered with $2^m, 2^{m+1}, \ldots, 2^{m+1}–1$.

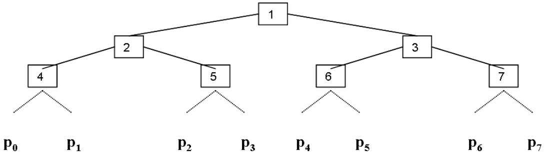

Figure 1: Tournament tree for $m = 2$

Tournament algorithm - description (continuation)

- Each processing unit competes to access a node of the tree.
- Initially, each processing unit competes to access a leaf of the tree.
- On each level, the winner gets the permission to climb on the upper level (parent $p(v)$ of node v for which it competed and won).
- Here it competes with the winner of the competition corresponding to the other child of $p(v)$.
- The processing unit that wins the competition for the root node enters the critical section.

Tournament algorithm - description (continuation)

- The shared variable $Priority[v]$ designates the processing unit that has the priority for the v node.
  - Initially, $Priority[v] = 0$; this means that the processing unit that competes for the v node from its left side has priority.
  - This variable is read and modified by both processing units that compete to access the v node.
- The shared variable *side* designates the side from which the processing unit that competes for the v node comes.
  - $side = 0$ means that the processing unit won the competition for the left child of the v node;
  - $side = 1$ means that the other processing unit won the competition for the right child of the v node.
- Each processing unit rises a flag ($Want[v][side] = 1$ or $Want[v][1 - side] = 1$) and then checks the flag of the other processing unit.
  - The processing unit that does not have priority, obeys the flag of the processing unit that has priority; if this is risen, it avoids to go to the upper level.
  - The processing unit that has priority, if it wishes, will advance to the upper level and will designate the other processing unit (the one it competed with) as having priority in the next phase. This ensures the property of *no-starvation* (*no-lockout*).
  - If the processing unit that has priority does not want to advance to the upper level, the other processing unit is free to advance to that level, if it wishes.

Tournament algorithm - pseudocode

Notations: Each node v is associated with three binary shared variables: $Priority[v]$, $Want[v][0]$ and $Want[v][1]$. Initially, the values of these variables are equal to 0.

Premise: In order to access the critical resource, p_i executes $\text{TOURNAMENT\_ENTRY\_EM}(2^m + \lfloor \frac{i}{2} \rfloor, i \bmod 2)$.

The pseudocode for processing unit p_i, $i \in \{0, 1, \ldots, n-1\}$, that competes to access node v from the position *side*.

$\langle Entry \rangle$ /*p_i needs the critical resource*/

$\text{TOURNAMENT\_ENTRY\_EM}(v, side)$

1 $Want[v][side] \leftarrow 0$ /* p_i's flag is down */
2 /* p_i waits until it has priority or until the adversary lowers the flag */
3 *wait until* $Priority[v] = side$ *or* $Want[v][1-side] = 0$
4 $Want[v][side] \leftarrow 1$ /* p_i rises the flag */
5 **if** $Priority[v] = 1 - side$
6   **then if** $Want[v][1-side] = 1$ /* if adversary has priority and its flag is up */
7   **then** *go to line* 1
8   **else** /* if adversary has no priority and adversary's flag is up */
9   *wait until* $Want[v][1-side] = 0$ /* wait until the adversaty lowers the flag */
10 **if** $v = 1$ /* if v is the root node */
11   **then** $\langle Critical\_Section \rangle$ /* p_i enters critical section */
12   **else** /* set $side = (v \bmod 2)$ and access the parent of node v */
13     $\text{TOURNAMENT\_ENTRY\_EM}(\lfloor v/2 \rfloor, v \bmod 2)$
14 $Priority[v] \leftarrow 1 - side$ /* p_i ceases its priority to the adversary */
15 $Want[v][side] \leftarrow 0$ /* p_i lowers the flag */

$\langle Exit \rangle$ /*p_i exits critical section*/ $\langle Remainder \rangle$ /*p_i does not need the critical resource*/

Correctness

Homework