Implementing Mutual Exclusion in Shared Memory

Peterson's and Bakery Algorithm

Fırat Ağış
firat@ceng.metu.edu.tr

Wireless Systems, Networks and Cybersecurity Laboratory
Department of Computer Engineering
Middle East Technical University
Ankara Turkey

May 12, 2024

Outline of the Presentation

Agenda

The problem

Race Conditions

Race conditions occur in concurrent systems when the outcome of a process depends on the timing or sequence of other events. This can lead to unpredictable behavior, data corruption, or system failures. Detecting and preventing race conditions is essential for ensuring the correctness and reliability of concurrent software systems.

Agenda

Mutual Exclusion

Mutual exclusion is the most common solution for race conditions in terms of mutual memory.

- SharedExclusion
  - Peterson's Algorithm
  - Bakery Algorithm

Agenda

Motivation/Importance

Disturbed systems with many process, while working on the same data, it is common for many processes to want to access or modify it at the same time. But a process changing the data while the other is reading it might create unwanted effects. Mutual exclusion is a common way of handling these effects. This enables multiple processes to work on the same piece of data, which is one of the most fundamental capabilities requested from a distributed system.

Agenda

Definitions

- When a process wants to modify a piece of data that is being accessed by others, it creates a **race condition**.
- Regions of code that modifies the shared memory, meaning regions that might cause race conditions are called the **critical section**.
- **Mutual exclusion** is the methodology of allowing only a single process to enter a critical section at a time.

Background

My comparisons are limited to the contemporaries of my algorithms.

- Dijkstra (1965) [1]
- Knuth (1966) [2]
- de Bruijn (1967) [3]
- Dijkstra (1968) [4]
- Eisenberg and McGuire (1972) [5]

Agenda

SharedExclusion Algorithm 1

Peterson's Algorithm

Peterson's Algorithm [6] prevents 2 processes from entering the critical section in the same time. It achieves this by a way of each process giving way to other. When the process comes to the critical section, it first allows the other process to continue to the critical section and only moves forward if the other is not currently trying to enter the critical section or if the other process finishes its work within the critical section. While it is a gentleman's way of handling shared memory exclusion, its main limitation lies in the fact that it only works for 2 processes.

SharedExclusion Algorithm 1

Peterson's Algorithm

Data: Processes p_i where $i \in \{0, 1\}$
initialization;
`turn` $\leftarrow$ **F**;
`waiting`[2];

algorithm;
if p_i *wants to enter* ***CS*** **then**
    `waiting`$[i] \leftarrow$ **T**;
    `turn` $\leftarrow 1 - i$;
    while $waiting[1 - i] = T$ *and* $turn = 1 - i$ **do** no-op;
    p_i enters **CS**;
    p_i exits **CS**;
    `waiting`$[i] \leftarrow$ **F**;
end

Algorithm 1: Peterson's algorithm

SharedExclusion Algorithm 2

Bakery Algorithm

Bakery Algorithm [7] simulates the activity of waiting in a line in its namesake to get your order. Every process takes a ticket with a number on it. When their number flashes on the screen, they come and give their order. If two processes have the same number on their ticket, the conflict is resolved through seniority, just like letting an elderly person go before you when both of you arrive at the same time.

SharedExclusion Algorithm 2

Bakery Algorithm

Data: Processes p_i where $i \in [1, n]$
initialization;
`entering`$[n]$;
`ticket`$[n]$;

algorithm;
if p_i *wants to enter* ***CS*** **then**
    `entering`$[i] \leftarrow$ **T**;
    `ticket`$[i] \leftarrow \max(\texttt{ticket}) + 1$;
    `entering`$[i] \leftarrow$ **F**;

    for $j \leftarrow 1$ **to** n **do**
        while $entering[j] = T$ **do** no-op;
        while $ticket[j] \neq 0$ *and* $(ticket[j], j) < (ticket[i], i)$ **do** no-op;
    end

    p_i enters **CS**;

    p_i exits **CS**;

    `ticket`$[i] \leftarrow 0$;
end

Algorithm 2: Bakery algorithm

Implementation Details

- Central vs Distributed Implementation.
- Non-event driven initialization.
- Leader election.

Agenda

Testing

- Peterson's Algorithm
  - Topology with two nodes.
- Bakery Algorithm
  - Well-connected topologies ($O(n^2)$ connections)
  - Topologies where only connections are between the leader and others.
  - Two directional ring topologies without shortcuts.
  - One-directional ring topologies without shortcuts.

Future Work

- Increase the event drivenness of the current implementation.
- Implement distributed versions of the algorithms and compare them to current implementations.

Agenda

Conclusions

What was achieved?

- We learned about mutual exclusion.
- Functional (but not at full potential) algorithms were added to the AHCv2 platform.
- I gained experience in event-driven and distributed computing.

References

[1] E. W. Dijkstra, “Solution of a problem in concurrent programming control,” *Commun. ACM*, vol. 8, no. 9, p. 569, sep 1965. [Online]. Available: https://doi.org/10.1145/365559.365617

[2] D. E. Knuth, “Additional comments on a problem in concurrent programming control,” *Commun. ACM*, vol. 9, no. 5, p. 321–322, may 1966. [Online]. Available: https://doi.org/10.1145/355592.365595

[3] N. G. de Bruijn, “Additional comments on a problem in concurrent programming control,” *Commun. ACM*, vol. 10, no. 3, p. 137–138, mar 1967. [Online]. Available: https://doi.org/10.1145/363162.363167

[4] E. W. Dijkstra, “The structure of the “the”-multiprogramming system,” *Commun. ACM*, vol. 11, no. 5, p. 341–346, may 1968. [Online]. Available: https://doi.org/10.1145/363095.363143

[5] M. A. Eisenberg and M. R. McGuire, “Further comments on dijkstra’s concurrent programming control problem,” *Commun. ACM*, vol. 15, no. 11, p. 999, nov 1972. [Online]. Available: https://doi.org/10.1145/355606.361895

[6] G. Peterson, “Myths about the mutual exclusion problem,” *Information Processing Letters*, vol. 12, no. 3, p. 115–116, Jun 1981.

[7] L. Lamport, “A new solution of dijkstra’s concurrent programming problem,” *Communications of the ACM*, vol. 17, no. 8, p. 453–455, Aug 1974.

Questions

THANK YOU

Implementing Mutual Exclusion in Shared Memory

Peterson's and Bakery Algorithm

presented by Fırat Ağış
firat@ceng.metu.edu.tr

WINS

May 12, 2024