# Distributed Algorithms
Lecture 2

Lecture by Dr. Gadi Taubenfeld
Typeset by Steven Karas

2017-04-05
Last edited 20:23:23 2017-04-05

## 1 Mutual Exclusion

First defined by Dijkstra in 1965.

**Critical Section** We describe the **critical section** as the portion of the code that uses a limited resource and should only be executed by one process at a time.

**Requirement: Mutual Exclusion** No two processes execute the critical section simultaneously.

**Requirement: Deadlock Freedom** If a process enters the locking code, at least one process (not necessarily the same one) must move on to the critical section eventually.

**Requirement: Starvation Freedom** If a process enters the locking code, it must eventually move onto the critical section.

All of these properties are closed under composition. When composing a starvation free function over a deadlock free function, the result is starvation free.

### 1.1 Peterson's Algorithm

Presented by Peterson in 1981.

```
Thread 0
flag[0] = true
turn = 1
while (flag[1] and turn = 1):
  sleep
CRITICAL SECTION
flag[0] = false
```

```
Thread 1
flag[1] = true
turn = 0
while (flag[0] and turn = 0):
  sleep
CRITICAL SECTION
flag[1] = false
```

This algorithm has the properties of mutual exclusion and starvation freedom.

**Safe Registers** Safe registers provide a relaxed model for memory access, where reads and writes are not atomic. Concurrent reads with writes can return any values that are valid during the read (all overlapping writes and the previous value).

Under this model, the algorithm still provides starvation freedom.

**Adaptation for single-writer registers** If we modify the multi-writer register *turn* such that it is two registers that are written by each thread, we can define a mapping such that the value is equivalent to 0 if they are equal and 1 if they are different.

**Contention free time complexity** 4 memory accesses in the preamble code. As an exercise, it may be possible to reduce this to 3.

**Process time complexity** Unbounded. More importantly, every two-process locking algorithm is unbound[1]

**Extending this to multiple processes** Use a tournament algorithm to provide $O(\log n)$ for contention free complexity.

# 2 Shared Memory Models

## 2.1 Simple shared memory

All processes access the shared memory in the same way with the same cost. May be atomic or only safe.

## 2.2 Coherent Caching

Each process has a cache, which may be marked as dirty. Accessing the cache is much cheaper than the shared memory.

## 2.3 Distributed Shared Memory

Each process has attached memory, and directly accesses other processes' memory. Peterson is very inefficient for this, but there exist some that are very good.

# 3 Computational Time Models

## 3.1 Synchronous

Extremely powerful, but unrealistic.

[1] Proof from 1992 in book section 3.2.5

### 3.2 Partially Synchronous

### 3.3 Asynchronous

Less powerful, but very realistic.

**Note** I left the lecture at this point.