ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2018 Prof. R. Wattenhofer

Computer Engineering II

Exercise Sheet Chapter 9

We categorize questions into four different categories:

Quiz Short questions which we will solve rather interactively at the start of the exercise sessions.

Basic Improve the basic understanding of the lecture material.

Advanced Test your ability to work with the lecture content. This is the typical style of questions which appear in the exam.

Mastery Beyond the essentials, more interesting, but also more challenging. These questions are **optional**, and we do not expect you to solve such exercises during the exam.

Questions marked with $^{(g)}$ may need some research on Google.

Quiz

1 Quiz (True or False)

a) The operation `counter++` is atomic.

b) Race conditions are difficult to debug because they appear seemingly at random.

c) Peterson's solution does not work on modern hardware because the memory accesses may be reordered to increase performance.

Advanced

2 Bad Peterson

In the script we mention that reordering the steps of Peterson's Algorithm can make the algorithm fail. Give a possible execution of two processes running Algorithm 1 (compared to the correct version of Peterson's Algorithm in the script, Lines 1 and 2 have been switched) such that both processes are in their critical section at the same time.

Algorithm 1 No Mutual Exclusion: Faulty Version of Peterson's Algorithm

Initialization: Shared registers W_0, W_1, Π, all initially 0.
Code for process p_i , $i = \{0, 1\}$
<**Entry**>
1: $\Pi := 1 - i$
2: $W_i := 1$
3: **repeat until** $\Pi = i$ or $W_{1-i} = 0$
<**Critical Section**>
4: ...
<**Exit**>
5: $W_i := 0$
<**Remainder Code**>
6: ...

3 Semaphores – Blocking or Spinning?

Algorithm 2 lock()

1: **repeat**
2: $r :=$ test-and-set(R)
3: **until** $r = 0$

Algorithm 3 unlock()

1: $R := 0$

Algorithm 4 Semaphore: wait()

Semaphore internals: integer S initialized to a non-negative value at creation of the semaphore, list of blocked processes L, register R
Input: process P that called wait()
<**Entry**>
1: lock(R) `// Algorithm 2`
<**Critical Section**>
2: **if** $S == 0$ **then**
3: L.addAsLast(P);
<**Exit**>
4: unlock(R)
5: P.block(); `// changes state of` P `to blocked`
6: **else**
7: S−−;
<**Exit**>
8: unlock(R) `// Algorithm 3`
9: **end if**

Algorithm 5 Semaphore: signal()

Semaphore internals: integer S initialized to a non-negative value at creation of the semaphore, list of blocked processes L, register R

<**Entry**>

1: lock(R) `// Algorithm 2`

<**Critical Section**>

2: **if** L is not empty **then**
3: $P = L$.removeFirst();
4: P.unblock(); `// changes state of P to ready`
5: **else**
6: S++;
7: **end if**

<**Exit**>

8: unlock(R) `// Algorithm 3`

Algorithms 4 and 5 implement a semaphore (same algorithms as in the script). This is one way of implementing a *blocking semaphore* since a wait()ing process blocks until the semaphore is positive. Another way of implementing a semaphore would be a *spinning semaphore* where instead of a wait()ing process blocking, it does busy waiting instead until it can lock the semaphore.

a) In which situations is a spinning semaphore more efficient than a blocking semaphore?

b) Wait, our implementation of a blocking semaphore does busy waiting too, that's how the algorithm to lock() a register (used in Algorithm 4 Line 1) was implemented (see Algorithm 2). Compare Algorithms 6 and 7. In Algorithm 6, we use our implementation of a blocking semaphore to synchronize processes. In Algorithm 7, we use a spinlock instead. Since our semaphore implementation uses a spinlock for its critical section, how is Algorithm 6 an improvement over Algorithm 7 in terms of running time efficiency?

Algorithm 6 Process using blocking semaphore

Internals: blocking semaphore `mutex` as implemented in Algorithms 4 and 5

<**Entry**>

1: mutex.wait() `// Algorithm 4`

<**Critical Section**>

2: ...

<**Exit**>

3: mutex.signal() `// Algorithm 5`

Algorithm 7 Process using test-and-set spinlock

Internals: register R

<**Entry**>

1: lock(R) `// Algorithm 2`

<**Critical Section**>

2: ...

<**Exit**>

3: unlock(R) `// Algorithm 3`

4 Lost Wakeup Problem

In the lecture we presented an implementation of a blocking semaphore (Algorithms 4 and 5).

a) Can Algorithm 4 Lines 4 and 5 be switched without a problem?

b) If Algorithm 4 Lines 4 and 5 are left in the order they are in, there is a scheduling issue that can occur that we did not mention in the lecture; the issue affects blocking and unblocking. Give an interleaving of the execution of two processes working on a binary semaphore that illustrates the problem. One process has to be running Algorithm 4 and the other Algorithm 5. (In the sample solutions, we will explain how this problem can be resolved.)

5 readers-Writers Problem (or Second Readers-Writers Problem)

In the script we gave a solution to the Readers-writers problem using semaphores. In this exercise, your task is to give a solution to the readers-Writers problem. The problem is this:

- Multiple readers have to be allowed to read simultaneously.
- While a process modifies the data no other process can be allowed to read or modify it.
- While a writer has requested write access, no process can be allowed to start reading.

Solve the problem using semaphores without a monitor.

6 Dentists' Office Problem

Imagine a dentists' office with multiple dentists, each with his own treatment room (Behandlungszimmer). When it opens up in the morning, the dentists manning the treatment rooms are still tired, so they sleep in their chairs. Whenever a customer comes in, he checks if there is a free dentist, and if so, he goes to the free treatment room and wakes up the dentist. If no dentist is free, the customers wait in the waiting room, which has a total of n chairs. If neither a dentist nor a waiting room chair are free, the customer leaves. If there are no customers waiting to be served, each free dentist goes to sleep.

Write pseudcode for a monitor as well as customers and dentists that solves the Dentists Problem.